OpenShift Container Platform 4.12

Scalability and performance

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

Last Updated: 2024-05-01

OpenShift Container Platform 4.12 Scalability and performance

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

Legal Notice

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 PERFORMANCE AND SCALABILITY PRACTICES 11
1.1. RECOMMENDED CONTROL PLANE PRACTICES 11
1.1.1. Recommended practices for scaling the cluster 11
1.1.2. Control plane node sizing 11
1.1.2.1. Selecting a larger Amazon Web Services instance type for control plane machines 14
1.1.2.1.1. Changing the Amazon Web Services instance type by using a control plane machine set 14
1.1.2.1.2. Changing the Amazon Web Services instance type by using the AWS console 15
1.2. RECOMMENDED INFRASTRUCTURE PRACTICES 16
1.2.1. Infrastructure node sizing 16
1.2.2. Scaling the Cluster Monitoring Operator 17
1.2.3. Prometheus database storage requirements 17
1.2.4. Configuring cluster monitoring 18
1.2.5. Additional resources 19
1.3. RECOMMENDED ETCD PRACTICES 19
1.3.1. Recommended etcd practices 19
1.3.2. Moving etcd to a different disk 21
1.3.3. Defragmenting etcd data 25
1.3.3.1. Automatic defragmentation 25
1.3.3.2. Manual defragmentation 26

CHAPTER 2. PLANNING YOUR ENVIRONMENT ACCORDING TO OBJECT MAXIMUMS 29
2.1. OPENSHIFT CONTAINER PLATFORM TESTED CLUSTER MAXIMUMS FOR MAJOR RELEASES 29
2.1.1. Example scenario 31
2.2. OPENSHIFT CONTAINER PLATFORM ENVIRONMENT AND CONFIGURATION ON WHICH THE CLUSTER MAXIMUMS ARE TESTED 32
2.2.1. AWS cloud platform 32
2.2.2. IBM Power platform 33
2.2.3. IBM Z platform 33
2.3. HOW TO PLAN YOUR ENVIRONMENT ACCORDING TO TESTED CLUSTER MAXIMUMS 34
2.4. HOW TO PLAN YOUR ENVIRONMENT ACCORDING TO APPLICATION REQUIREMENTS 35

CHAPTER 3. RECOMMENDED HOST PRACTICES FOR IBM Z & IBM(R) LINUXONE ENVIRONMENTS 38
3.1. MANAGING CPU OVERCOMMITMENT 38
3.2. DISABLE TRANSPARENT HUGE PAGES 38
3.3. BOOST NETWORKING PERFORMANCE WITH RECEIVE FLOW STEERING 39
3.3.1. Use the Machine Config Operator (MCO) to activate RFS 39
3.4. CHOOSE YOUR NETWORKING SETUP 40
3.5. ENSURE HIGH DISK PERFORMANCE WITH HYPERPAV ON Z/VM 40
3.5.1. Use the Machine Config Operator (MCO) to activate HyperPAV aliases in nodes using z/VM full-pack minidisks 41
3.6. RHEL KVM ON IBM Z HOST RECOMMENDATIONS 42
3.6.1. Use I/O threads for your virtual block devices 42
3.6.2. Avoid virtual SCSI devices 43
3.6.3. Configure guest caching for disk 43
3.6.4. Exclude the memory balloon device 43
3.6.5. Tune the CPU migration algorithm of the host scheduler 43
3.6.6. Disable the cpuset cgroup controller 44
3.6.7. Tune the polling period for idle virtual CPUs 44

CHAPTER 4. USING THE NODE TUNING OPERATOR 46
4.1. ABOUT THE NODE TUNING OPERATOR 46
4.2. ACCESSING AN EXAMPLE NODE TUNING OPERATOR SPECIFICATION 46

4.3. DEFAULT PROFILES SET ON A CLUSTER 47
4.4. VERIFYING THAT THE TUNED PROFILES ARE APPLIED 48
4.5. CUSTOM TUNING SPECIFICATION 48
4.6. CUSTOM TUNING EXAMPLES 53
4.7. SUPPORTED TUNED DAEMON PLUGINS 54
4.8. CONFIGURING NODE TUNING IN A HOSTED CLUSTER 55
4.9. ADVANCED NODE TUNING FOR HOSTED CLUSTERS BY SETTING KERNEL BOOT PARAMETERS 58

CHAPTER 5. USING CPU MANAGER AND TOPOLOGY MANAGER 62
5.1. SETTING UP CPU MANAGER 62
5.2. TOPOLOGY MANAGER POLICIES 66
5.3. SETTING UP TOPOLOGY MANAGER 67
5.4. POD INTERACTIONS WITH TOPOLOGY MANAGER POLICIES 67

CHAPTER 6. SCHEDULING NUMA-AWARE WORKLOADS 69
6.1. ABOUT NUMA-AWARE SCHEDULING 69
6.2. INSTALLING THE NUMA RESOURCES OPERATOR 70
6.2.1. Installing the NUMA Resources Operator using the CLI 70
6.2.2. Installing the NUMA Resources Operator using the web console 72
6.3. SCHEDULING NUMA-AWARE WORKLOADS 73
6.3.1. Creating the NUMAResourcesOperator custom resource 73
6.3.2. Deploying the NUMA-aware secondary pod scheduler 74
6.3.3. Scheduling workloads with the NUMA-aware scheduler 76
6.4. SCHEDULING NUMA-AWARE WORKLOADS WITH MANUAL PERFORMANCE SETTINGS 79
6.4.1. Creating the NUMAResourcesOperator custom resource with manual performance settings 79
6.4.2. Deploying the NUMA-aware secondary pod scheduler with manual performance settings 81
6.4.3. Scheduling workloads with the NUMA-aware scheduler with manual performance settings 83
6.5. OPTIONAL: CONFIGURING POLLING OPERATIONS FOR NUMA RESOURCES UPDATES 87
6.6. TROUBLESHOOTING NUMA-AWARE SCHEDULING 88
6.6.1. Checking the NUMA-aware scheduler logs 92
6.6.2. Troubleshooting the resource topology exporter 94
6.6.3. Correcting a missing resource topology exporter config map 95

CHAPTER 7. SCALABILITY AND PERFORMANCE OPTIMIZATION 98
7.1. OPTIMIZING STORAGE 98
7.1.1. Available persistent storage options 98
7.1.2. Recommended configurable storage technology 99
7.1.2.1. Specific application storage recommendations 100
7.1.2.1.1. Registry 100
7.1.2.1.2. Scaled registry 100
7.1.2.1.3. Metrics 100
7.1.2.1.4. Logging 101
7.1.2.1.5. Applications 101
7.1.2.2. Other specific application storage recommendations 101
7.1.3. Data storage management 102
7.1.4. Optimizing storage performance for Microsoft Azure 103
7.1.5. Additional resources 103
7.2. OPTIMIZING ROUTING 103
7.2.1. Baseline Ingress Controller (router) performance 103
7.2.2. Configuring Ingress Controller liveness, readiness, and startup probes 105
7.2.3. Configuring HAProxy reload interval 106
7.3. OPTIMIZING NETWORKING 106
7.3.1. Optimizing the MTU for your network 107
7.3.2. Recommended practices for installing large scale clusters 107

7.3.3. Impact of IPsec 108
7.3.4. Additional resources 108
7.4. OPTIMIZING CPU USAGE WITH MOUNT NAMESPACE ENCAPSULATION 108
7.4.1. Encapsulating mount namespaces 108
7.4.2. Configuring mount namespace encapsulation 111
7.4.3. Inspecting encapsulated namespaces 113
7.4.4. Running additional services in the encapsulated namespace 114
7.4.5. Additional resources 114

CHAPTER 8. MANAGING BARE METAL HOSTS 116
8.1. ABOUT BARE METAL HOSTS AND NODES 116
8.2. MAINTAINING BARE METAL HOSTS 116
8.2.1. Adding a bare metal host to the cluster using the web console 116
8.2.2. Adding a bare metal host to the cluster using YAML in the web console 117
8.2.3. Automatically scaling machines to the number of available bare metal hosts 118
8.2.4. Removing bare metal hosts from the provisioner node 119

CHAPTER 9. MONITORING BARE-METAL EVENTS WITH THE BARE METAL EVENT RELAY 121
9.1. ABOUT BARE-METAL EVENTS 121
9.2. HOW BARE-METAL EVENTS WORK 121
9.2.1. Bare Metal Event Relay data flow 122
9.2.1.1. Operator-managed pod 122
9.2.1.2. Bare Metal Event Relay 122
9.2.1.3. Cloud native event 122
9.2.1.4. CNCF CloudEvents 122
9.2.1.5. HTTP transport or AMQP dispatch router 122
9.2.1.6. Cloud event proxy sidecar 123
9.2.2. Redfish message parsing service 123
9.2.3. Installing the Bare Metal Event Relay using the CLI 123
9.2.4. Installing the Bare Metal Event Relay using the web console 124
9.3. INSTALLING THE AMQ MESSAGING BUS 125
9.4. SUBSCRIBING TO REDFISH BMC BARE-METAL EVENTS FOR A CLUSTER NODE 126
9.4.1. Subscribing to bare-metal events 126
9.4.2. Querying Redfish bare-metal event subscriptions with curl 129
9.4.3. Creating the bare-metal event and Secret CRs 130
9.5. SUBSCRIBING APPLICATIONS TO BARE-METAL EVENTS REST API REFERENCE 132
api/ocloudNotifications/v1/subscriptions 132
HTTP method 132
Description 132
HTTP method 133
Description 133
api/ocloudNotifications/v1/subscriptions/<subscription_id> 133
HTTP method 133
Description 133
api/ocloudNotifications/v1/health/ 134
HTTP method 134
Description 134
9.6. MIGRATING CONSUMER APPLICATIONS TO USE HTTP TRANSPORT FOR PTP OR BARE-METAL EVENTS 134

CHAPTER 10. WHAT HUGE PAGES DO AND HOW THEY ARE CONSUMED BY APPLICATIONS 136
10.1. WHAT HUGE PAGES DO 136
10.2. HOW HUGE PAGES ARE CONSUMED BY APPS 136
10.3. CONSUMING HUGE PAGES RESOURCES USING THE DOWNWARD API 137

10.4. CONFIGURING HUGE PAGES AT BOOT TIME 139
10.5. DISABLING TRANSPARENT HUGE PAGES 141

CHAPTER 11. LOW LATENCY TUNING 142
11.1. UNDERSTANDING LOW LATENCY 142
11.1.1. About hyperthreading for low latency and real-time applications 143
11.2. PROVISIONING REAL-TIME AND LOW LATENCY WORKLOADS 143
11.2.1. Known limitations for real-time 144
11.2.2. Provisioning a worker with real-time capabilities 144
11.2.3. Verifying the real-time kernel installation 146
11.2.4. Creating a workload that works in real-time 146
11.2.5. Creating a pod with a QoS class of Guaranteed 146
11.2.6. Optional: Disabling CPU load balancing for DPDK 148
11.2.7. Assigning a proper node selector 148
11.2.8. Scheduling a workload onto a worker with real-time capabilities 149
11.2.9. Reducing power consumption by taking CPUs offline 149
11.2.10. Optional: Power saving configurations 150
11.2.11. Managing device interrupt processing for guaranteed pod isolated CPUs 152
11.2.11.1. Disabling CPU CFS quota 152
11.2.11.2. Disabling global device interrupts handling in Node Tuning Operator 153
11.2.11.3. Disabling interrupt processing for individual pods 153
11.2.12. Upgrading the performance profile to use device interrupt processing 153
11.2.12.1. Supported API Versions 153
11.2.12.1.1. Upgrading Node Tuning Operator API from v1alpha1 to v1 154
11.2.12.1.2. Upgrading Node Tuning Operator API from v1alpha1 or v1 to v2 154
11.3. TUNING NODES FOR LOW LATENCY WITH THE PERFORMANCE PROFILE 154
11.3.1. Configuring huge pages 155
11.3.2. Allocating multiple huge page sizes 156
11.3.3. Configuring a node for IRQ dynamic load balancing 156
11.3.4. About support of IRQ affinity setting 159
11.3.5. Configuring hyperthreading for a cluster 160
11.3.5.1. Disabling hyperthreading for low latency applications 162
11.3.6. Understanding workload hints 163
11.3.7. Configuring workload hints manually 164
11.3.8. Restricting CPUs for infra and application containers 165
11.4. REDUCING NIC QUEUES USING THE NODE TUNING OPERATOR 167
11.4.1. Adjusting the NIC queues with the performance profile 167
11.4.2. Verifying the queue status 170
11.4.3. Logging associated with adjusting NIC queues 173
11.5. DEBUGGING LOW LATENCY CNF TUNING STATUS 174
11.5.1. Machine config pools 175
11.6. COLLECTING LOW LATENCY TUNING DEBUGGING DATA FOR RED HAT SUPPORT 176
11.6.1. About the must-gather tool 176
11.6.2. About collecting low latency tuning data 177
11.6.3. Gathering data about specific features 177

CHAPTER 12. PERFORMING LATENCY TESTS FOR PLATFORM VERIFICATION 179
12.1. PREREQUISITES FOR RUNNING LATENCY TESTS 179
12.2. ABOUT DISCOVERY MODE FOR LATENCY TESTS 179
Limiting the nodes used during tests 179
12.3. MEASURING LATENCY 180
12.4. RUNNING THE LATENCY TESTS 181
12.4.1. Running hwlatdetect 182

Example hwlatdetect test results 184
12.4.2. Running cyclictest 186
Example cyclictest results 187
12.4.3. Running oslat 189
12.5. GENERATING A LATENCY TEST FAILURE REPORT 190
12.6. GENERATING A JUNIT LATENCY TEST REPORT 191
12.7. RUNNING LATENCY TESTS ON A SINGLE-NODE OPENSHIFT CLUSTER 191
12.8. RUNNING LATENCY TESTS IN A DISCONNECTED CLUSTER 192
Mirroring the images to a custom registry accessible from the cluster 192
Configuring the tests to consume images from a custom registry 193
Mirroring images to the cluster OpenShift image registry 193
Mirroring a different set of test images 194
12.9. TROUBLESHOOTING ERRORS WITH THE CNF-TESTS CONTAINER 195

CHAPTER 13. IMPROVING CLUSTER STABILITY IN HIGH LATENCY ENVIRONMENTS USING WORKER LATENCY PROFILES 196
13.1. UNDERSTANDING WORKER LATENCY PROFILES 196
13.2. IMPLEMENTING WORKER LATENCY PROFILES AT CLUSTER CREATION 199
13.3. USING AND CHANGING WORKER LATENCY PROFILES 200
13.4. EXAMPLE STEPS FOR DISPLAYING RESULTING VALUES OF WORKERLATENCYPROFILE 202

CHAPTER 14. CREATING A PERFORMANCE PROFILE 204
14.1. ABOUT THE PERFORMANCE PROFILE CREATOR 204
14.1.1. Gathering data about your cluster using the must-gather command 204
14.1.2. Running the Performance Profile Creator using podman 205
14.1.2.1. How to run podman to create a performance profile 208
14.1.3. Running the Performance Profile Creator wrapper script 209
14.1.4. Performance Profile Creator arguments 213
14.2. REFERENCE PERFORMANCE PROFILES 216
14.2.1. A performance profile template for clusters that use OVS-DPDK on OpenStack 216
14.3. ADDITIONAL RESOURCES 217

CHAPTER 15. WORKLOAD PARTITIONING IN SINGLE-NODE OPENSHIFT 218

CHAPTER 16. REQUESTING CRI-O AND KUBELET PROFILING DATA BY USING THE NODE OBSERVABILITY OPERATOR 219
16.1. WORKFLOW OF THE NODE OBSERVABILITY OPERATOR 219
16.2. INSTALLING THE NODE OBSERVABILITY OPERATOR 219
16.2.1. Installing the Node Observability Operator using the CLI 219
16.2.2. Installing the Node Observability Operator using the web console 221
16.3. CREATING THE NODE OBSERVABILITY CUSTOM RESOURCE 221
16.4. RUNNING THE PROFILING QUERY 223

CHAPTER 17. CLUSTERS AT THE NETWORK FAR EDGE 225
17.1. CHALLENGES OF THE NETWORK FAR EDGE 225
17.1.1. Overcoming the challenges of the network far edge 225
17.1.2. Using ZTP to provision clusters at the network far edge 226
17.1.3. Installing managed clusters with SiteConfig resources and RHACM 227
17.1.4. Configuring managed clusters with policies and PolicyGenTemplate resources 228
17.2. PREPARING THE HUB CLUSTER FOR ZTP 230
17.2.1. Telco RAN 4.12 validated solution software versions 230
17.2.2. Installing GitOps ZTP in a disconnected environment 230
17.2.3. Adding RHCOS ISO and RootFS images to the disconnected mirror host 231
17.2.4. Enabling the assisted service 232

17.2.5. Configuring the hub cluster to use a disconnected mirror registry 233
17.2.6. Configuring the hub cluster to use unauthenticated registries 235
17.2.7. Configuring the hub cluster with ArgoCD 236
17.2.8. Preparing the GitOps ZTP site configuration repository 238
17.3. INSTALLING MANAGED CLUSTERS WITH RHACM AND SITECONFIG RESOURCES 239
17.3.1. GitOps ZTP and Topology Aware Lifecycle Manager 240
17.3.2. Overview of deploying managed clusters with ZTP 241
Overview of the managed site installation process 242
17.3.3. Creating the managed bare-metal host secrets 242
17.3.4. Configuring Discovery ISO kernel arguments for installations using GitOps ZTP 243
17.3.5. Deploying a managed cluster with SiteConfig and ZTP 245
17.3.5.1. Single-node OpenShift SiteConfig CR installation reference 249
17.3.6. Monitoring managed cluster installation progress 251
17.3.7. Troubleshooting GitOps ZTP by validating the installation CRs 251
17.3.8. Troubleshooting {ztp} virtual media booting on Supermicro servers 252
17.3.9. Removing a managed cluster site from the ZTP pipeline 253
17.3.10. Removing obsolete content from the ZTP pipeline 253
17.3.11. Tearing down the ZTP pipeline 254
17.4. CONFIGURING MANAGED CLUSTERS WITH POLICIES AND POLICYGENTEMPLATE RESOURCES 254
17.4.1. About the PolicyGenTemplate CRD 254
17.4.2. Recommendations when customizing PolicyGenTemplate CRs 258
17.4.3. PolicyGenTemplate CRs for RAN deployments 258
17.4.4. Customizing a managed cluster with PolicyGenTemplate CRs 260
17.4.5. Monitoring managed cluster policy deployment progress 261
17.4.6. Validating the generation of configuration policy CRs 263
17.4.7. Restarting policy reconciliation 264
17.4.8. Changing applied managed cluster CRs using policies 265
17.4.9. Indication of done for ZTP installations 267
17.5. MANUALLY INSTALLING A SINGLE-NODE OPENSHIFT CLUSTER WITH ZTP 268
17.5.1. Generating ZTP installation and configuration CRs manually 268
17.5.2. Creating the managed bare-metal host secrets 273
17.5.3. Configuring Discovery ISO kernel arguments for manual installations using GitOps ZTP 274
17.5.4. Installing a single managed cluster 275
17.5.5. Monitoring the managed cluster installation status 276
17.5.6. Troubleshooting the managed cluster 277
17.5.7. RHACM generated cluster installation CRs reference 278
17.6. RECOMMENDED SINGLE-NODE OPENSHIFT CLUSTER CONFIGURATION FOR VDU APPLICATION WORKLOADS 280
17.6.1. Running low latency applications on OpenShift Container Platform 280
17.6.2. Recommended cluster host requirements for vDU application workloads 281
17.6.3. Configuring host firmware for low latency and high performance 281
17.6.4. Connectivity prerequisites for managed cluster networks 282
17.6.5. Workload partitioning in single-node OpenShift with GitOps ZTP 283
17.6.6. Recommended installation-time cluster configurations 283
17.6.6.1. Workload partitioning 283
17.6.6.2. Reduced platform management footprint 285
17.6.6.3. SCTP 287
17.6.6.4. Accelerated container startup 288
17.6.6.5. Automatic kernel crash dumps with kdump 292
17.6.7. Recommended postinstallation cluster configurations 294
17.6.7.1. Operator namespaces and Operator groups 294
17.6.7.2. Operator subscriptions 296
17.6.7.3. Cluster logging and log forwarding 297

17.6.7.4. Performance profile 298
17.6.7.5. PTP 299
17.6.7.6. Extended Tuned profile 302
17.6.7.7. SR-IOV 302
17.6.7.8. Console Operator 304
17.6.7.9. Alertmanager 304
17.6.7.10. Operator Lifecycle Manager 305
17.6.7.11. Network diagnostics 305
17.7. VALIDATING SINGLE-NODE OPENSHIFT CLUSTER TUNING FOR VDU APPLICATION WORKLOADS 306
17.7.1. Recommended firmware configuration for vDU cluster hosts 306
17.7.2. Recommended cluster configurations to run vDU applications 307
17.7.2.1. Recommended cluster MachineConfig CRs 308
17.7.2.2. Recommended cluster Operators 308
17.7.2.3. Recommended cluster kernel configuration 309
17.7.2.4. Checking the realtime kernel version 309
17.7.3. Checking that the recommended cluster configurations are applied 310
17.8. ADVANCED MANAGED CLUSTER CONFIGURATION WITH SITECONFIG RESOURCES 320
17.8.1. Customizing extra installation manifests in the ZTP GitOps pipeline 320
17.8.2. Filtering custom resources using SiteConfig filters 321
17.9. ADVANCED MANAGED CLUSTER CONFIGURATION WITH POLICYGENTEMPLATE RESOURCES 322
17.9.1. Deploying additional changes to clusters 322
17.9.2. Using PolicyGenTemplate CRs to override source CRs content 323
17.9.3. Adding custom content to the GitOps ZTP pipeline 326
17.9.4. Configuring policy compliance evaluation timeouts for PolicyGenTemplate CRs 328
17.9.5. Signalling ZTP cluster deployment completion with validator inform policies 330
17.9.6. Configuring PTP events with PolicyGenTemplate CRs 331
17.9.6.1. Configuring PTP events that use HTTP transport 331
17.9.6.2. Configuring PTP events that use AMQP transport 333
17.9.7. Configuring bare-metal events with PolicyGenTemplate CRs 335
17.9.7.1. Configuring bare-metal events that use HTTP transport 335
17.9.7.2. Configuring bare-metal events that use AMQP transport 336
17.9.8. Configuring the Image Registry Operator for local caching of images 338
17.9.8.1. Configuring disk partitioning with SiteConfig 338
17.9.8.2. Configuring the image registry using PolicyGenTemplate CRs 339
17.9.9. Using hub templates in PolicyGenTemplate CRs 342
17.9.9.1. Example hub templates 343
17.9.9.2. Specifying host NICs in site PolicyGenTemplate CRs with hub cluster templates 344
17.9.9.3. Specifying VLAN IDs in group PolicyGenTemplate CRs with hub cluster templates 346
17.9.9.4. Syncing new ConfigMap changes to existing PolicyGenTemplate CRs 347
17.10. UPDATING MANAGED CLUSTERS WITH THE TOPOLOGY AWARE LIFECYCLE MANAGER 348
17.10.1. About the Topology Aware Lifecycle Manager configuration 349
17.10.2. About managed policies used with Topology Aware Lifecycle Manager 349
17.10.3. Installing the Topology Aware Lifecycle Manager by using the web console 350
17.10.4. Installing the Topology Aware Lifecycle Manager by using the CLI 350
17.10.5. About the ClusterGroupUpgrade CR 351
17.10.5.1. Selecting clusters 352
17.10.5.2. Validating 355
17.10.5.3. Pre-caching 355
17.10.5.4. Creating a backup 355
17.10.5.5. Updating clusters 356
17.10.5.6. Update status 358
17.10.5.7. Blocking ClusterGroupUpgrade CRs 361

17.10.6. Update policies on managed clusters 367
17.10.6.1. Configuring Operator subscriptions for managed clusters that you install with TALM 369
17.10.6.2. Applying update policies to managed clusters 369
17.10.7. Creating a backup of cluster resources before upgrade 376
17.10.7.1. Creating a ClusterGroupUpgrade CR with backup 377
17.10.7.2. Recovering a cluster after a failed upgrade 379
17.10.8. Using the container image pre-cache feature 382
17.10.8.1. Creating a ClusterGroupUpgrade CR with pre-caching 383
17.10.9. Troubleshooting the Topology Aware Lifecycle Manager 385
17.10.9.1. General troubleshooting 386
17.10.9.2. Cannot modify the ClusterUpgradeGroup CR 386
17.10.9.3. Managed policies 387
Checking managed policies on the system 387
Checking remediationAction mode 387
Checking policy compliance state 387
17.10.9.4. Clusters 388
Checking if managed clusters are present 388
Checking if managed clusters are available 389
Checking clusterLabelSelector 389
Checking if canary clusters are present 390
Checking the pre-caching status on spoke clusters 390
17.10.9.5. Remediation Strategy 391
Checking if remediationStrategy is present in the ClusterGroupUpgrade CR 391
Checking if maxConcurrency is specified in the ClusterGroupUpgrade CR 391
17.10.9.6. Topology Aware Lifecycle Manager 391
Checking condition message and status in the ClusterGroupUpgrade CR 391
Checking corresponding copied policies 391
Checking if status.remediationPlan was computed 392
Errors in the TALM manager container 392
Clusters are not compliant to some policies after a ClusterGroupUpgrade CR has completed 393
17.11. UPDATING MANAGED CLUSTERS IN A DISCONNECTED ENVIRONMENT WITH THE TOPOLOGY AWARE LIFECYCLE MANAGER 393
17.11.1. Updating clusters in a disconnected environment 393
17.11.1.1. Setting up the environment 393
17.11.1.2. Performing a platform update 395
17.11.1.3. Performing an Operator update 398
17.11.1.3.1. Troubleshooting missed Operator updates due to out-of-date policy compliance states 403
17.11.1.4. Performing a platform and an Operator update together 404
17.11.1.5. Removing Performance Addon Operator subscriptions from deployed clusters 407
17.11.2. About the auto-created ClusterGroupUpgrade CR for ZTP 408
17.12. UPDATING GITOPS ZTP 409
17.12.1. Overview of the GitOps ZTP update process 410
17.12.2. Preparing for the upgrade 410
17.12.3. Labeling the existing clusters 411
17.12.4. Stopping the existing GitOps ZTP applications 411
17.12.5. Required changes to the Git repository 412
17.12.6. Installing the new GitOps ZTP applications 413
17.12.7. Rolling out the GitOps ZTP configuration changes 414
17.13. EXPANDING SINGLE-NODE OPENSHIFT CLUSTERS WITH GITOPS ZTP 414
17.13.1. Applying profiles to the worker node 415
17.13.2. (Optional) Ensuring PTP and SR-IOV daemon selector compatibility 415
17.13.3. PTP and SR-IOV node selector compatibility 416
17.13.4. Using PolicyGenTemplate CRs to apply worker node policies to worker nodes 417

17.13.5. Adding worker nodes to single-node OpenShift clusters with GitOps ZTP 419
17.14. PRE-CACHING IMAGES FOR SINGLE-NODE OPENSHIFT DEPLOYMENTS 422
17.14.1. Getting the factory-precaching-cli tool 423
17.14.2. Booting from a live operating system image 423
17.14.3. Partitioning the disk 425
17.14.3.1. Creating the partition 426
17.14.3.2. Mounting the partition 427
17.14.4. Downloading the images 428
17.14.4.1. Downloading with parallel workers 428
17.14.4.2. Preparing to download the OpenShift Container Platform images 429
17.14.4.3. Downloading the OpenShift Container Platform images 430
17.14.4.4. Downloading the Operator images 432
17.14.4.5. Pre-caching custom images in disconnected environments 433
17.14.5. Pre-caching images in ZTP 437
17.14.5.1. Understanding the clusters.ignitionConfigOverride field 440
17.14.5.2. Understanding the nodes.installerArgs field 440
17.14.5.3. Understanding the nodes.ignitionConfigOverride field 441
17.14.6. Troubleshooting 442
17.14.6.1. Rendered catalog is invalid 442

CHAPTER 1. RECOMMENDED PERFORMANCE AND SCALABILITY PRACTICES

1.1. RECOMMENDED CONTROL PLANE PRACTICES

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

1.1.1. Recommended practices for scaling the cluster

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

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

When scaling up the cluster to higher node counts:

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

NOTE

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

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

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

NOTE

When scaling large and dense clusters to lower node counts, it might take large amounts of time because the process involves draining or evicting the objects running on the nodes being terminated in parallel. Also, the client might start to throttle the requests if there are too many objects to evict. The default client queries per second (QPS) and burst rates are currently set to **5** and **10** respectively. These values cannot be modified in OpenShift Container Platform.

1.1.2. Control plane node sizing

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

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

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

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

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

IMPORTANT

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

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

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

IMPORTANT

You can modify the control plane node size in a running OpenShift Container Platform 4.12 cluster for the following configurations only:

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

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

IMPORTANT

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

NOTE

In OpenShift Container Platform 4.12, 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.

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

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

NOTE

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

If you are uncertain about the state of the **ControlPlaneMachineSet** CR in your cluster, you can verify the CR status.

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

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

Prerequisites

- Your AWS cluster uses a control plane machine set.

Procedure

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

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

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

```
providerSpec:
  value:
    ...
    instanceType: <compatible_aws_instance_type> 1
```

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

3. Save your changes.

    - For clusters that use the default **RollingUpdate** update strategy, the Operator automatically propagates the changes to your control plane configuration.

    - For clusters that are configured to use the **OnDelete** update strategy, you must replace your control plane machines manually.

Additional resources

- Managing control plane machines with control plane machine sets

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

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

Prerequisites

- You have access to the AWS console with the permissions required to modify the EC2 Instance for your cluster.

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

Procedure

1. Open the AWS console and fetch the instances for the control plane machines.

2. Choose one control plane machine instance.

    a. For the selected control plane machine, back up the etcd data by creating an etcd snapshot. For more information, see "Backing up etcd".

    b. In the AWS console, stop the control plane machine instance.

    c. Select the stopped instance, and click **Actions → Instance Settings → Change instance type**.

    d. 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**.

    e. Start the instance.

f. If your OpenShift Container Platform cluster has a corresponding **Machine** object for the instance, update the instance type of the object to match the instance type set in the AWS console.

3. Repeat this process for each control plane machine.

Additional resources

- Backing up etcd
- AWS documentation about changing the instance type

1.2. RECOMMENDED INFRASTRUCTURE PRACTICES

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

1.2.1. Infrastructure node sizing

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

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

In general, three infrastructure nodes are recommended per cluster.

IMPORTANT

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

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

NOTE

In OpenShift Container Platform 4.12, 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.

1.2.2. 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**.

1.2.3. Prometheus database storage requirements

Red Hat performed various tests for different scale sizes.

NOTE

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

Table 1.1. Prometheus Database storage requirements based on number of nodes/pods in the cluster

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

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

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

NOTE

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

Recommendations for OpenShift Container Platform

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

1.2.4. Configuring cluster monitoring

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

Procedure

To increase the storage capacity for Prometheus:

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

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

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

② ④ The storage class for your cluster.

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

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

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

3. Save the file.

4. Apply the changes by running:

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

1.2.5. Additional resources

- Infrastructure Nodes in OpenShift 4
- OpenShift Container Platform cluster maximums
- Creating infrastructure machine sets

1.3. RECOMMENDED ETCD PRACTICES

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

1.3.1. 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 an OpenShift API slowness, which affects cluster performance. Because of these reasons, avoid colocating other workloads on the control-plane nodes that are I/O sensitive or intensive and share the same underlying I/O infrastructure.

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 10ms, 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.

NOTE

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

The following hard 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.

NOTE

Avoid NAS or SAN setups and spinning drives. Ceph Rados Block Device (RBD) and other types of network-attached storage can result in unpredictable network latency. To provide fast storage to etcd nodes at scale, use PCI passthrough to pass NVM devices directly to the nodes.

Always benchmark by using utilities such as fio. You can use such utilities to continuously monitor the cluster performance as it increases.

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/cloud-bulldozer/etcd-perf
```

- If you use Docker, run this command:

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

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

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

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

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

Additional resources

- How to use **fio** to check etcd disk performance in OpenShift Container Platform
- etcd performance troubleshooting guide for OpenShift Container Platform

1.3.2. Moving etcd to a different disk

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

The Machine Config Operator (MCO) is responsible for mounting a secondary disk for OpenShift Container Platform 4.12 container storage.

NOTE

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

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have access to the cluster with **cluster-admin** privileges.
- The **MachineConfigPool** must match **metadata.labels[machineconfiguration.openshift.io/role]**. This applies to a controller, worker, or a custom pool.

Procedure

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

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

   ```
   # lsblk
   ```

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

2. Create a **MachineConfig** YAML file named **etcd-mc.yml** with contents such as the following, replacing instances of **<new_disk_name>** with the noted device name:

   ```
   apiVersion: machineconfiguration.openshift.io/v1
   kind: MachineConfig
   metadata:
     labels:
       machineconfiguration.openshift.io/role: master
     name: 98-var-lib-etcd
   spec:
     config:
       ignition:
         version: 3.2.0
       systemd:
         units:
         - contents: |
             [Unit]
             Description=Make File System on /dev/<new_disk_name>
             DefaultDependencies=no
             BindsTo=dev-<new_disk_name>.device
             After=dev-<new_disk_name>.device var.mount
             Before=systemd-fsck@dev-<new_disk_name>.service

             [Service]
             Type=oneshot
             RemainAfterExit=yes
             ExecStart=/usr/lib/systemd/systemd-makefs xfs /dev/<new_disk_name>
             TimeoutSec=0

             [Install]
             WantedBy=var-lib-containers.mount
           enabled: true
           name: systemd-mkfs@dev-<new_disk_name>.service
         - contents: |
             [Unit]
             Description=Mount /dev/<new_disk_name> to /var/lib/etcd
             Before=local-fs.target
             Requires=systemd-mkfs@dev-<new_disk_name>.service
             After=systemd-mkfs@dev-<new_disk_name>.service var.mount

             [Mount]
             What=/dev/<new_disk_name>
             Where=/var/lib/etcd
             Type=xfs
   ```

```
          Options=defaults,prjquota

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

          [Service]
          Type=oneshot
          RemainAfterExit=yes
          ExecCondition=/usr/bin/test ! -d /var/lib/etcd/member
          ExecStart=semanage fcontext -a -e /sysroot/ostree/deploy/rhcos/var/lib/etcd/
/var/lib/etcd/
          ExecStart=/bin/rsync -ar /sysroot/ostree/deploy/rhcos/var/lib/etcd/ /var/lib/etcd/
          TimeoutSec=0

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

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

          [Install]
          WantedBy=multi-user.target graphical.target
        enabled: true
        name: restorecon-var-lib-etcd.service
```

3. Log in to the cluster as a user with **cluster-admin** privileges and create the machine configuration:

```
$ oc login -u <username> -p <password>
```

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

The nodes are updated and rebooted. After the reboot completes, the following events occur:

- An XFS file system is created on the specified disk.

- The disk mounts to **/var/lib/etcd**.
- The content from **/sysroot/ostree/deploy/rhcos/var/lib/etcd** syncs to **/var/lib/etcd**.
- A restore of **SELinux** labels is forced for **/var/lib/etcd**.
- The old content is not removed.

4. After the nodes are on a separate disk, update the **etcd-mc.yml** file with contents such as the following, replacing instances of **<new_disk_name>** with the noted device name:

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: 98-var-lib-etcd
spec:
  config:
    ignition:
      version: 3.2.0
    systemd:
      units:
      - contents: |
          [Unit]
          Description=Mount /dev/<new_disk_name> to /var/lib/etcd
          Before=local-fs.target
          Requires=systemd-mkfs@dev-<new_disk_name>.service
          After=systemd-mkfs@dev-<new_disk_name>.service var.mount

          [Mount]
          What=/dev/<new_disk_name>
          Where=/var/lib/etcd
          Type=xfs
          Options=defaults,prjquota

          [Install]
          WantedBy=local-fs.target
        enabled: true
        name: var-lib-etcd.mount
```

5. Apply the modified version that removes the logic for creating and syncing the device to prevent the nodes from rebooting:

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

Verification steps

- Run the **grep <new_disk_name> /proc/mounts** command in a debug shell for the node to ensure that the disk mounted:

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

```
# grep <new_disk_name> /proc/mounts
```

Example output

```
/dev/nvme1n1 /var/lib/etcd xfs
rw,seclabel,relatime,attr2,inode64,logbufs=8,logbsize=32k,prjquota 0 0
```

Additional resources

- Red Hat Enterprise Linux CoreOS (RHCOS)

1.3.3. Defragmenting etcd data

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

Monitor these key metrics:

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

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

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

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

NOTE

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

1.3.3.1. 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>
```

1.3.3.2. Manual defragmentation

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

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

You can also determine whether defragmentation is needed by checking the etcd database size in MB that will be freed by defragmentation with the PromQL expression: **(etcd_mvcc_db_total_size_in_bytes - etcd_mvcc_db_total_size_in_use_in_bytes)/1024/1024**

WARNING

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

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

Prerequisites

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

Procedure

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

a. Get the list of etcd pods:

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

Example output

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

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

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

Example output

```
Defaulting container name to etcdctl.
Use 'oc describe pod/etcd-ip-10-0-159-225.example.redhat.com -n openshift-etcd' to see
all of the containers in this pod.
+---------------------------+------------------+---------+---------+-----------+------------+-----------
+------------+--------------------+--------+
|         ENDPOINT          |        ID        | VERSION | DB SIZE | IS LEADER | IS LEARNER |
RAFT TERM | RAFT INDEX | RAFT APPLIED INDEX | ERRORS |
+---------------------------+------------------+---------+---------+-----------+------------+-----------
+------------+--------------------+--------+
|  https://10.0.191.37:2379 | 251cd44483d811c3 |   3.4.9 |  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
```

CHAPTER 2. PLANNING YOUR ENVIRONMENT ACCORDING TO OBJECT MAXIMUMS

Consider the following tested object maximums when you plan your OpenShift Container Platform cluster.

These guidelines are based on the largest possible cluster. For smaller clusters, the maximums are lower. There are many factors that influence the stated thresholds, including the etcd version or storage data format.

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

WARNING

Clusters that experience rapid change, such as those with many starting and stopping pods, can have a lower practical maximum size than documented.

2.1. OPENSHIFT CONTAINER PLATFORM TESTED CLUSTER MAXIMUMS FOR MAJOR RELEASES

NOTE

Red Hat does not provide direct guidance on sizing your OpenShift Container Platform cluster. This is because determining whether your cluster is within the supported bounds of OpenShift Container Platform requires careful consideration of all the multidimensional factors that limit the cluster scale.

OpenShift Container Platform supports tested cluster maximums rather than absolute cluster maximums. Not every combination of OpenShift Container Platform version, control plane workload, and network plugin are tested, so the following table does not represent an absolute expectation of scale for all deployments. It might not be possible to scale to a maximum on all dimensions simultaneously. The table contains tested maximums for specific workload and deployment configurations, and serves as a scale guide as to what can be expected with similar deployments.

Maximum type	4.x tested maximum
Number of nodes	2,000 [1]
Number of pods [2]	150,000
Number of pods per node	500 [3]
Number of pods per core	There is no default value.

Maximum type	4.x tested maximum
Number of namespaces [4]	10,000
Number of builds	10,000 (Default pod RAM 512 Mi) - Source-to-Image (S2I) build strategy
Number of pods per namespace [5]	25,000
Number of routes and back ends per Ingress Controller	2,000 per router
Number of secrets	80,000
Number of config maps	90,000
Number of services [6]	10,000
Number of services per namespace	5,000
Number of back-ends per service	5,000
Number of deployments per namespace [5]	2,000
Number of build configs	12,000
Number of custom resource definitions (CRD)	512 [7]

1. Pause pods were deployed to stress the control plane components of OpenShift Container Platform at 2000 node scale. The ability to scale to similar numbers will vary depending upon specific deployment and workload parameters.

2. The pod count displayed here is the number of test pods. The actual number of pods depends on the application's memory, CPU, and storage requirements.

3. This was tested on a cluster with 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 keyspace grows excessively large and exceeds the space quota. Periodic maintenance of etcd, including defragmentation, is highly recommended to free etcd storage.

5. There are 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.

2.1.1. Example scenario

As an example, 500 worker nodes (m5.2xl) were tested, and are supported, using OpenShift Container Platform 4.12, the OVN-Kubernetes network plugin, and the following workload objects:

- 200 namespaces, in addition to the defaults
- 60 pods per node; 30 server and 30 client pods (30k total)
- 57 image streams/ns (11.4k total)
- 15 services/ns backed by the server pods (3k total)
- 15 routes/ns backed by the previous services (3k total)
- 20 secrets/ns (4k total)
- 10 config maps/ns (2k total)
- 6 network policies/ns, including deny-all, allow-from ingress and intra-namespace rules
- 57 builds/ns

The following factors are known to affect cluster workload scaling, positively or negatively, and should be factored into the scale numbers when planning a deployment. For additional information and guidance, contact your sales representative or Red Hat support.

- Number of pods per node
- Number of containers per pod
- Type of probes used (for example, liveness/readiness, exec/http)
- Number of network policies
- Number of projects, or namespaces
- Number of image streams per project
- Number of builds per project
- Number of services/endpoints and type

- Number of routes
- Number of shards
- Number of secrets
- Number of config maps
- Rate of API calls, or the cluster "churn", which is an estimation of how quickly things change in the cluster configuration.
  - Prometheus query for pod creation requests per second over 5 minute windows: **sum(irate(apiserver_request_count{resource="pods",verb="POST"}[5m]))**
  - Prometheus query for all API requests per second over 5 minute windows: **sum(irate(apiserver_request_count{}[5m]))**
- Cluster node resource consumption of CPU
- Cluster node resource consumption of memory

2.2. OPENSHIFT CONTAINER PLATFORM ENVIRONMENT AND CONFIGURATION ON WHICH THE CLUSTER MAXIMUMS ARE TESTED

2.2.1. 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.

2.2.2. 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.

2.2.3. 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 instanced. Next, a network and CPU demanding client/server workload were used to evaluate the stability of the system under stress. Client and server pods were pairwise deployed and each pair was spread over two compute nodes.

3. No separate workload node was used. The workload simulates a microservice workload between two compute nodes.

4. Physical number of processors used is six Integrated Facilities for Linux (IFLs).

5. Total physical memory used is 512 GiB.

2.3. 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:

```
required pods per cluster / pods per node = total number of nodes needed
```

The default maximum number of pods per node is 250. However, the number of pods that fit on a node is dependent on the application itself. Consider the application’s memory, CPU, and storage requirements, as described in "How to plan your environment according to application requirements".

Example scenario

If you want to scope your cluster for 2200 pods per cluster, you would need at least five nodes, assuming that there are 500 maximum pods per node:

```
2200 / 500 = 4.4
```

If you increase the number of nodes to 20, then the pod distribution changes to 110 pods per node:

```
2200 / 20 = 110
```

Where:

> required pods per cluster / total number of nodes = expected pods per node

OpenShift Container Platform comes with several system pods, such as SDN, DNS, Operators, and others, which run across every worker node by default. Therefore, the result of the above formula can vary.

2.4. HOW TO PLAN YOUR ENVIRONMENT ACCORDING TO APPLICATION REQUIREMENTS

Consider an example application environment:

Pod type	Pod quantity	Max memory	CPU cores	Persistent storage
apache	100	500 MB	0.5	1 GB
node.js	200	1 GB	1	1 GB
postgresql	100	1 GB	2	10 GB
JBoss EAP	100	1 GB	1	1 GB

Extrapolated requirements: 550 CPU cores, 450GB RAM, and 1.4TB storage.

Instance size for nodes can be modulated up or down, depending on your preference. Nodes are often resource overcommitted. In this deployment scenario, you can choose to run additional smaller nodes or fewer larger nodes to provide the same amount of resources. Factors such as operational agility and cost-per-instance should be considered.

Node type	Quantity	CPUs	RAM (GB)
Nodes (option 1)	100	4	16
Nodes (option 2)	50	8	32
Nodes (option 3)	25	16	64

Some applications lend themselves well to overcommitted environments, and some do not. Most Java applications and applications that use huge pages are examples of applications that would not allow for overcommitment. That memory can not be used for other applications. In the example above, the environment would be roughly 30 percent overcommitted, a common ratio.

The application pods can access a service either by using environment variables or DNS. If using environment variables, for each active service the variables are injected by the kubelet when a pod is run on a node. A cluster-aware DNS server watches the Kubernetes API for new services and creates a set of DNS records for each one. If DNS is enabled throughout your cluster, then all pods should automatically be able to resolve services by their DNS name. Service discovery using DNS can be used in case you must go beyond 5000 services. When using environment variables for service discovery, the argument list exceeds the allowed length after 5000 services in a namespace, then the pods and deployments will start failing. Disable the service links in the deployment's service specification file to overcome this:

```
---
apiVersion: template.openshift.io/v1
kind: Template
metadata:
  name: deployment-config-template
  creationTimestamp:
  annotations:
    description: This template will create a deploymentConfig with 1 replica, 4 env vars and a service.
    tags: ''
objects:
- apiVersion: apps.openshift.io/v1
  kind: DeploymentConfig
  metadata:
    name: deploymentconfig${IDENTIFIER}
  spec:
    template:
      metadata:
        labels:
          name: replicationcontroller${IDENTIFIER}
      spec:
        enableServiceLinks: false
        containers:
        - name: pause${IDENTIFIER}
          image: "${IMAGE}"
          ports:
          - containerPort: 8080
            protocol: TCP
          env:
          - name: ENVVAR1_${IDENTIFIER}
            value: "${ENV_VALUE}"
          - name: ENVVAR2_${IDENTIFIER}
            value: "${ENV_VALUE}"
          - name: ENVVAR3_${IDENTIFIER}
            value: "${ENV_VALUE}"
          - name: ENVVAR4_${IDENTIFIER}
            value: "${ENV_VALUE}"
          resources: {}
          imagePullPolicy: IfNotPresent
          capabilities: {}
          securityContext:
            capabilities: {}
            privileged: false
        restartPolicy: Always
        serviceAccount: ''
    replicas: 1
    selector:
      name: replicationcontroller${IDENTIFIER}
    triggers:
    - type: ConfigChange
    strategy:
      type: Rolling
- apiVersion: v1
```

```
  kind: Service
  metadata:
    name: service${IDENTIFIER}
  spec:
    selector:
      name: replicationcontroller${IDENTIFIER}
    ports:
    - name: serviceport${IDENTIFIER}
      protocol: TCP
      port: 80
      targetPort: 8080
    clusterIP: ''
    type: ClusterIP
    sessionAffinity: None
  status:
    loadBalancer: {}
parameters:
- name: IDENTIFIER
  description: Number to append to the name of resources
  value: '1'
  required: true
- name: IMAGE
  description: Image to use for deploymentConfig
  value: gcr.io/google-containers/pause-amd64:3.0
  required: false
- name: ENV_VALUE
  description: Value to use for environment variables
  generate: expression
  from: "[A-Za-z0-9]{255}"
  required: false
labels:
  template: deployment-config-template
```

The number of application pods that can run in a namespace is dependent on the number of services and the length of the service name when the environment variables are used for service discovery. **ARG_MAX** on the system defines the maximum argument length for a new process and it is set to 2097152 bytes (2 MiB) by default. The Kubelet injects environment variables in to each pod scheduled to run in the namespace including:

- **<SERVICE_NAME>_SERVICE_HOST=<IP>**
- **<SERVICE_NAME>_SERVICE_PORT=<PORT>**
- **<SERVICE_NAME>_PORT=tcp://<IP>:<PORT>**
- **<SERVICE_NAME>_PORT_<PORT>_TCP=tcp://<IP>:<PORT>**
- **<SERVICE_NAME>_PORT_<PORT>_TCP_PROTO=tcp**
- **<SERVICE_NAME>_PORT_<PORT>_TCP_PORT=<PORT>**
- **<SERVICE_NAME>_PORT_<PORT>_TCP_ADDR=<ADDR>**

The pods in the namespace will start to fail if the argument length exceeds the allowed value and the number of characters in a service name impacts it. For example, in a namespace with 5000 services, the limit on the service name is 33 characters, which enables you to run 5000 pods in the namespace.

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

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

NOTE

The s390x architecture is unique in many aspects. Therefore, some recommendations made here might not apply to other platforms.

NOTE

Unless stated otherwise, these practices apply to both z/VM and Red Hat Enterprise Linux (RHEL) KVM installations on IBM Z and IBM® LinuxONE.

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 IBM® LinuxONE environments.

3.6.1. Use I/O threads for your virtual block devices

To make virtual block devices use I/O threads, you must configure one or more I/O threads for the virtual server and each virtual block device to use one of these I/O threads.

The following example specifies **<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 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.

3.6.2. Avoid virtual SCSI devices

Configure virtual SCSI devices only if you need to address the device through SCSI-specific interfaces. Configure disk space as virtual block devices rather than virtual SCSI devices, regardless of the backing on the host.

However, you might need SCSI-specific interfaces for:

- A LUN for a SCSI-attached tape drive on the host.
- A DVD ISO file on the host file system that is mounted on a virtual DVD drive.

3.6.3. Configure guest caching for disk

Configure your disk devices to do caching by the guest and not by the host.

Ensure that the driver element of the disk device includes the **cache="none"** and **io="native"** parameters.

```
<disk type="block" device="disk">
    <driver name="qemu" type="raw" cache="none" io="native" iothread="1"/>
...
</disk>
```

3.6.4. Exclude the memory balloon device

Unless you need a dynamic memory size, do not define a memory balloon device and ensure that libvirt does not create one for you. Include the **memballoon** parameter as a child of the devices element in your domain configuration XML file.

- Check the list of active profiles:

  ```
  <memballoon model="none"/>
  ```

3.6.5. Tune the CPU migration algorithm of the host scheduler

IMPORTANT

Do not change the scheduler settings unless you are an expert who understands the implications. Do not apply changes to production systems without testing them and confirming that they have the intended effect.

The **kernel.sched_migration_cost_ns** parameter specifies a time interval in nanoseconds. After the last execution of a task, the CPU cache is considered to have useful content until this interval expires. Increasing this interval results in fewer task migrations. The default value is 500000 ns.

If the CPU idle time is higher than expected when there are runnable processes, try reducing this interval. If tasks bounce between CPUs or nodes too often, try increasing it.

To dynamically set the interval to 60000 ns, enter the following command:

```
# sysctl kernel.sched_migration_cost_ns=60000
```

To persistently change the value to 60000 ns, add the following entry to **/etc/sysctl.conf**:

```
kernel.sched_migration_cost_ns=60000
```

3.6.6. Disable the cpuset cgroup controller

NOTE

This setting applies only to KVM hosts with cgroups version 1. To enable CPU hotplug on the host, disable the cgroup controller.

Procedure

1. Open **/etc/libvirt/qemu.conf** with an editor of your choice.

2. Go to the **cgroup_controllers** line.

3. Duplicate the entire line and remove the leading number sign (#) from the copy.

4. Remove the **cpuset** entry, as follows:

   ```
   cgroup_controllers = [ "cpu", "devices", "memory", "blkio", "cpuacct" ]
   ```

5. For the new setting to take effect, you must restart the libvirtd daemon:

   a. Stop all virtual machines.

   b. Run the following command:

      ```
      # systemctl restart libvirtd
      ```

   c. Restart the virtual machines.

This setting persists across host reboots.

3.6.7. Tune the polling period for idle virtual CPUs

When a virtual CPU becomes idle, KVM polls for wakeup conditions for the virtual CPU before allocating the host resource. You can specify the time interval, during which polling takes place in sysfs at **/sys/module/kvm/parameters/halt_poll_ns**. During the specified time, polling reduces the wakeup latency for the virtual CPU at the expense of resource usage. Depending on the workload, a longer or shorter time for polling can be beneficial. The time interval is specified in nanoseconds. The default is 50000 ns.

- To optimize for low CPU consumption, enter a small value or write 0 to disable polling:

  ```
  # echo 0 > /sys/module/kvm/parameters/halt_poll_ns
  ```

- To optimize for low latency, for example for transactional workloads, enter a large value:

  ```
  # echo 80000 > /sys/module/kvm/parameters/halt_poll_ns
  ```

Additional resources

- Linux on IBM Z Performance Tuning for KVM
- Getting started with virtualization on IBM Z

CHAPTER 4. 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.

4.1. 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.

NOTE

Currently, disabling CPU load balancing is not supported by cgroup v2. As a result, you might not get the desired behavior from performance profiles if you have cgroup v2 enabled. Enabling cgroup v2 is not recommended if you are using performance profiles.

The Node Tuning Operator is part of a standard OpenShift Container Platform installation in version 4.1 and later.

NOTE

In earlier versions of OpenShift Container Platform, the Performance Addon Operator was used to implement automatic tuning to achieve low latency performance for OpenShift applications. In OpenShift Container Platform 4.11 and later, this functionality is part of the Node Tuning Operator.

4.2. ACCESSING AN EXAMPLE NODE TUNING OPERATOR SPECIFICATION

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

Procedure

Procedure

- Run the following command to access an example Node Tuning Operator specification:

```
oc get tuned.tuned.openshift.io/default -o yaml -n openshift-cluster-node-tuning-operator
```

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

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

4.3. DEFAULT PROFILES SET ON A CLUSTER

The following are the default profiles set on a cluster.

```
apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
  name: default
  namespace: openshift-cluster-node-tuning-operator
spec:
  profile:
  - data: |
      [main]
      summary=Optimize systems running OpenShift (provider specific parent profile)
      include=-provider-${f:exec:cat:/var/lib/tuned/provider},openshift
    name: openshift
  recommend:
  - profile: openshift-control-plane
    priority: 30
    match:
    - label: node-role.kubernetes.io/master
    - label: node-role.kubernetes.io/infra
  - profile: openshift-node
    priority: 40
```

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

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

4.4. VERIFYING THAT THE TUNED PROFILES ARE APPLIED

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

```
$ oc get profile.tuned.openshift.io -n openshift-cluster-node-tuning-operator
```

Example output

```
NAME             TUNED                     APPLIED   DEGRADED   AGE
master-0         openshift-control-plane   True      False      6h33m
master-1         openshift-control-plane   True      False      6h33m
master-2         openshift-control-plane   True      False      6h33m
worker-a         openshift-node            True      False      6h28m
worker-b         openshift-node            True      False      6h28m
```

- **NAME**: Name of the Profile object. There is one Profile object per node and their names match.
- **TUNED**: Name of the desired TuneD profile to apply.
- **APPLIED**: **True** if the TuneD daemon applied the desired profile. (**True**/**False**/**Unknown**).
- **DEGRADED**: **True** if any errors were reported during application of the TuneD profile (**True**/**False**/**Unknown**).
- **AGE**: Time elapsed since the creation of Profile object.

4.5. CUSTOM TUNING SPECIFICATION

The custom resource (CR) for the Operator has two major sections. The first section, **profile:**, is a list of TuneD profiles and their names. The second, **recommend:**, defines the profile selection logic.

Multiple custom tuning specifications can co-exist as multiple CRs in the Operator's namespace. The existence of new CRs or the deletion of old CRs is detected by the Operator. All existing custom tuning specifications are merged and appropriate objects for the containerized TuneD daemons are updated.

Management state

The Operator Management state is set by adjusting the default Tuned CR. By default, the Operator is in the Managed state and the **spec.managementState** field is not present in the default Tuned CR. Valid values for the Operator Management state are as follows:

- Managed: the Operator will update its operands as configuration resources are updated
- Unmanaged: the Operator will ignore changes to the configuration resources
- Removed: the Operator will remove its operands and resources the Operator provisioned

Profile data

The **profile:** section lists TuneD profiles and their names.

```
profile:
- name: tuned_profile_1
  data: |
    # TuneD profile specification
    [main]
    summary=Description of tuned_profile_1 profile

    [sysctl]
    net.ipv4.ip_forward=1
    # ... other sysctl's or other TuneD daemon plugins supported by the containerized TuneD

# ...

- name: tuned_profile_n
  data: |
    # TuneD profile specification
    [main]
    summary=Description of tuned_profile_n profile

    # tuned_profile_n profile settings
```

Recommended profiles

The **profile:** selection logic is defined by the **recommend:** section of the CR. The **recommend:** section is a list of items to recommend the profiles based on a selection criteria.

```
recommend:
<recommend-item-1>
# ...
<recommend-item-n>
```

The individual items of the list:

```
- machineConfigLabels: 1
    <mcLabels> 2
  match: 3
    <match> 4
  priority: <priority> 5
  profile: <tuned_profile_name> 6
  operand: 7
    debug: <bool> 8
    tunedConfig:
      reapply_sysctl: <bool> 9
```

1 Optional.

2 A dictionary of key/value **MachineConfig** labels. The keys must be unique.

3 If omitted, profile match is assumed unless a profile with a higher priority matches first or **machineConfigLabels** is set.

4 An optional list.

5 Profile ordering priority. Lower numbers mean higher priority (**0** is the highest priority).

6 A TuneD profile to apply on a match. For example **tuned_profile_1**.

7 Optional operand configuration.

8 Turn debugging on or off for the TuneD daemon. Options are **true** for on or **false** for off. The default is **false**.

9 Turn **reapply_sysctl** functionality on or off for the TuneD daemon. Options are **true** for on and **false** for off.

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

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

1 Node or pod label name.

2 Optional node or pod label value. If omitted, the presence of **<label_name>** is enough to match.

3 Optional object type (**node** or **pod**). If omitted, **node** is assumed.

4 An optional **<match>** list.

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

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

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

IMPORTANT

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

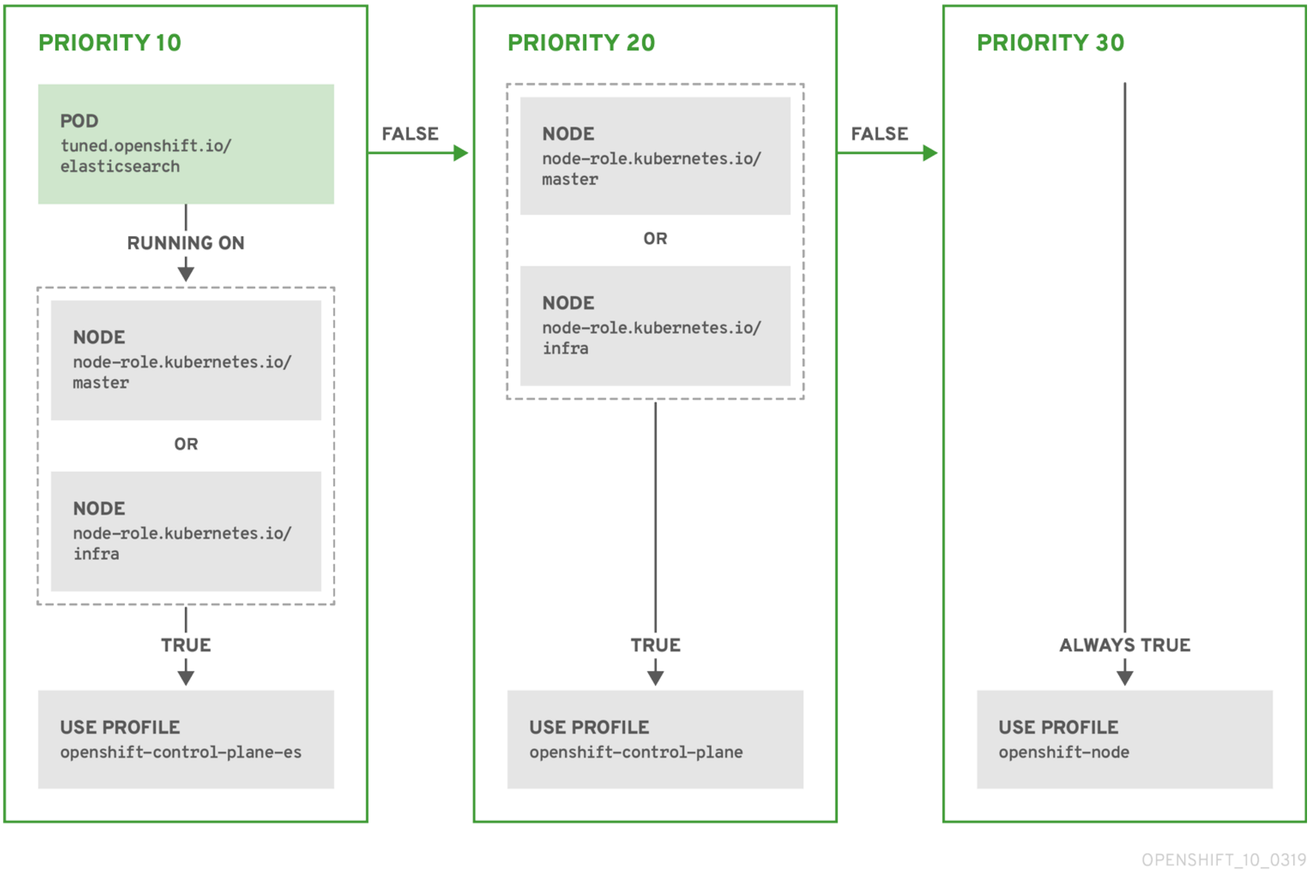
Example: node or pod label based matching

```
- match:
    - label: tuned.openshift.io/elasticsearch
      match:
      - label: node-role.kubernetes.io/master
      - label: node-role.kubernetes.io/infra
      type: pod
    priority: 10
    profile: openshift-control-plane-es
  - match:
    - label: node-role.kubernetes.io/master
    - label: node-role.kubernetes.io/infra
    priority: 20
    profile: openshift-control-plane
  - priority: 30
    profile: openshift-node
```

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

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

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



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.

4.6. 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.

4.7. SUPPORTED TUNED DAEMON PLUGINS

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

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

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

- script
- systemd

NOTE

The TuneD bootloader plugin only supports Red Hat Enterprise Linux CoreOS (RHCOS) worker nodes.

Additional resources

- Available TuneD Plugins
- Getting Started with TuneD

4.8. CONFIGURING NODE TUNING IN A HOSTED CLUSTER

IMPORTANT

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

For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

To set node-level tuning on the nodes in your hosted cluster, you can use the Node Tuning Operator. In hosted control planes, you can configure node tuning by creating config maps that contain **Tuned** objects and referencing those config maps in your node pools.

Procedure

1. Create a config map that contains a valid tuned manifest, and reference the manifest in a node pool. In the following example, a **Tuned** manifest defines a profile that sets **vm.dirty_ratio** to 55 on nodes that contain the **tuned-1-node-label** node label with any value. Save the following **ConfigMap** manifest in a file named **tuned-1.yaml**:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: tuned-1
  namespace: clusters
data:
  tuning: |
    apiVersion: tuned.openshift.io/v1
    kind: Tuned
    metadata:
      name: tuned-1
      namespace: openshift-cluster-node-tuning-operator
    spec:
      profile:
      - data: |
          [main]
          summary=Custom OpenShift profile
          include=openshift-node
          [sysctl]
          vm.dirty_ratio="55"
        name: tuned-1-profile
      recommend:
      - priority: 20
        profile: tuned-1-profile
```

NOTE

If you do not add any labels to an entry in the **spec.recommend** section of the Tuned spec, node-pool-based matching is assumed, so the highest priority profile in the **spec.recommend** section is applied to nodes in the pool. Although you can achieve more fine-grained node-label-based matching by setting a label value in the Tuned **.spec.recommend.match** section, node labels will not persist during an upgrade unless you set the **.spec.management.upgradeType** value of the node pool to **InPlace**.

2. Create the **ConfigMap** object in the management cluster:

```
$ oc --kubeconfig="$MGMT_KUBECONFIG" create -f tuned-1.yaml
```

3. Reference the **ConfigMap** object in the **spec.tuningConfig** field of the node pool, either by editing a node pool or creating one. In this example, assume that you have only one **NodePool**, named **nodepool-1**, which contains 2 nodes.

```
apiVersion: hypershift.openshift.io/v1alpha1
kind: NodePool
metadata:
  ...
  name: nodepool-1
  namespace: clusters
...
spec:
  ...
  tuningConfig:
  - name: tuned-1
status:
...
```

NOTE

You can reference the same config map in multiple node pools. In hosted control planes, the Node Tuning Operator appends a hash of the node pool name and namespace to the name of the Tuned CRs to distinguish them. Outside of this case, do not create multiple TuneD profiles of the same name in different Tuned CRs for the same hosted cluster.

Verification

Now that you have created the **ConfigMap** object that contains a **Tuned** manifest and referenced it in a **NodePool**, the Node Tuning Operator syncs the **Tuned** objects into the hosted cluster. You can verify which **Tuned** objects are defined and which TuneD profiles are applied to each node.

1. List the **Tuned** objects in the hosted cluster:

```
$ oc --kubeconfig="$HC_KUBECONFIG" get tuned.tuned.openshift.io -n openshift-cluster-node-tuning-operator
```

Example output

```
NAME      AGE
```

```
default    7m36s
rendered   7m36s
tuned-1    65s
```

2. List the **Profile** objects in the hosted cluster:

```
$ oc --kubeconfig="$HC_KUBECONFIG" get profile.tuned.openshift.io -n openshift-cluster-node-tuning-operator
```

Example output

```
NAME                    TUNED          APPLIED   DEGRADED   AGE
nodepool-1-worker-1          tuned-1-profile  True      False      7m43s
nodepool-1-worker-2          tuned-1-profile  True      False      7m14s
```

NOTE

If no custom profiles are created, the **openshift-node** profile is applied by default.

3. To confirm that the tuning was applied correctly, start a debug shell on a node and check the sysctl values:

```
$ oc --kubeconfig="$HC_KUBECONFIG" debug node/nodepool-1-worker-1 -- chroot /host sysctl vm.dirty_ratio
```

Example output

```
vm.dirty_ratio = 55
```

4.9. ADVANCED NODE TUNING FOR HOSTED CLUSTERS BY SETTING KERNEL BOOT PARAMETERS

IMPORTANT

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

For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

For more advanced tuning in hosted control planes, which requires setting kernel boot parameters, you can also use the Node Tuning Operator. The following example shows how you can create a node pool with huge pages reserved.

Procedure

1. Create a **ConfigMap** object that contains a **Tuned** object manifest for creating 10 huge pages that are 2 MB in size. Save this **ConfigMap** manifest in a file named **tuned-hugepages.yaml**:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: tuned-hugepages
  namespace: clusters
data:
  tuning: |
    apiVersion: tuned.openshift.io/v1
    kind: Tuned
    metadata:
      name: hugepages
      namespace: openshift-cluster-node-tuning-operator
    spec:
      profile:
      - data: |
          [main]
          summary=Boot time configuration for hugepages
          include=openshift-node
          [bootloader]
          cmdline_openshift_node_hugepages=hugepagesz=2M hugepages=50
        name: openshift-node-hugepages
      recommend:
      - priority: 20
        profile: openshift-node-hugepages
```

NOTE

The **.spec.recommend.match** field is intentionally left blank. In this case, this **Tuned** object is applied to all nodes in the node pool where this **ConfigMap** object is referenced. Group nodes with the same hardware configuration into the same node pool. Otherwise, TuneD operands can calculate conflicting kernel parameters for two or more nodes that share the same node pool.

2. Create the **ConfigMap** object in the management cluster:

```
$ oc --kubeconfig="$MGMT_KUBECONFIG" create -f tuned-hugepages.yaml
```

3. Create a **NodePool** manifest YAML file, customize the upgrade type of the **NodePool**, and reference the **ConfigMap** object that you created in the **spec.tuningConfig** section. Create the **NodePool** manifest and save it in a file named **hugepages-nodepool.yaml** by using the **hypershift** CLI:

```
NODEPOOL_NAME=hugepages-example
INSTANCE_TYPE=m5.2xlarge
NODEPOOL_REPLICAS=2

hypershift create nodepool aws \
  --cluster-name $CLUSTER_NAME \
  --name $NODEPOOL_NAME \
    --node-count $NODEPOOL_REPLICAS \
    --instance-type $INSTANCE_TYPE \
    --render > hugepages-nodepool.yaml
```

4. In the **hugepages-nodepool.yaml** file, set **.spec.management.upgradeType** to **InPlace**, and set **.spec.tuningConfig** to reference the **tuned-hugepages ConfigMap** object that you created.

```
apiVersion: hypershift.openshift.io/v1alpha1
kind: NodePool
metadata:
  name: hugepages-nodepool
  namespace: clusters
  ...
spec:
  management:
    ...
    upgradeType: InPlace
  ...
  tuningConfig:
  - name: tuned-hugepages
```

NOTE

To avoid the unnecessary re-creation of nodes when you apply the new **MachineConfig** objects, set **.spec.management.upgradeType** to **InPlace**. If you use the **Replace** upgrade type, nodes are fully deleted and new nodes can replace them when you apply the new kernel boot parameters that the TuneD operand calculated.

5. Create the **NodePool** in the management cluster:

```
$ oc --kubeconfig="$MGMT_KUBECONFIG" create -f hugepages-nodepool.yaml
```

Verification

After the nodes are available, the containerized TuneD daemon calculates the required kernel boot parameters based on the applied TuneD profile. After the nodes are ready and reboot once to apply the generated **MachineConfig** object, you can verify that the TuneD profile is applied and that the kernel boot parameters are set.

1. List the **Tuned** objects in the hosted cluster:

```
$ oc --kubeconfig="$HC_KUBECONFIG" get tuned.tuned.openshift.io -n openshift-cluster-node-tuning-operator
```

Example output

```
NAME                 AGE
default              123m
hugepages-8dfb1fed   1m23s
rendered             123m
```

2. List the **Profile** objects in the hosted cluster:

```
$ oc --kubeconfig="$HC_KUBECONFIG" get profile.tuned.openshift.io -n openshift-cluster-
node-tuning-operator
```

Example output

```
NAME                        TUNED                    APPLIED   DEGRADED   AGE
nodepool-1-worker-1         openshift-node           True      False      132m
nodepool-1-worker-2         openshift-node           True      False      131m
hugepages-nodepool-worker-1 openshift-node-hugepages True      False      4m8s
hugepages-nodepool-worker-2 openshift-node-hugepages True      False      3m57s
```

Both of the worker nodes in the new **NodePool** have the **openshift-node-hugepages** profile applied.

3. To confirm that the tuning was applied correctly, start a debug shell on a node and check **/proc/cmdline**.

```
$ oc --kubeconfig="$HC_KUBECONFIG" debug node/nodepool-1-worker-1 -- chroot /host
cat /proc/cmdline
```

Example output

```
BOOT_IMAGE=(hd0,gpt3)/ostree/rhcos-... hugepagesz=2M hugepages=50
```

Additional resources

For more information about hosted control planes, see Hosted control planes for Red Hat OpenShift Container Platform (Technology Preview).

CHAPTER 5. USING CPU MANAGER AND TOPOLOGY MANAGER

CPU Manager manages groups of CPUs and constrains workloads to specific CPUs.

CPU Manager is useful for workloads that have some of these attributes:

- Require as much CPU time as possible.
- Are sensitive to processor cache misses.
- Are low-latency network applications.
- Coordinate with other processes and benefit from sharing a single processor cache.

Topology Manager collects hints from the CPU Manager, Device Manager, and other Hint Providers to align pod resources, such as CPU, SR-IOV 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 the collected hints to decide if a pod can be accepted or rejected on a node, based on the configured Topology Manager policy and pod resources requested.

Topology Manager is useful for workloads that use hardware accelerators to support latency-critical execution and high throughput parallel computation.

To use Topology Manager you must configure CPU Manager with the **static** policy.

5.1. 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. This is the default policy.
- **static**. This policy allows containers in guaranteed pods with integer CPU requests. It also limits access to exclusive CPUs on the node. If **static**, you must use a lowercase **s**.

2 Optional. Specify the CPU Manager reconcile frequency. The default is **5s**.

5. Create the dynamic kubelet config:

```
# 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 **cpuManagerPolicy** is defined when you create the **KubeletConfig** CR.

② **cpuManagerReconcilePeriod** is defined when you create the **KubeletConfig** CR.

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

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

Example output

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

9. Create the pod:

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

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

```
# oc describe pod cpumanager
```

Example output

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

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

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

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

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

Example output

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

12. Check the allowed CPU list for the task:

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

Example output

```
 Cpus_allowed_list:    1
```

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

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

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

Example output

```
...
Capacity:
 attachable-volumes-aws-ebs:  39
 cpu:                         2
 ephemeral-storage:           124768236Ki
 hugepages-1Gi:          0
 hugepages-2Mi:          0
 memory:                 8162900Ki
 pods:                   250
Allocatable:
 attachable-volumes-aws-ebs:  39
 cpu:                    1500m
 ephemeral-storage:      124768236Ki
 hugepages-1Gi:          0
 hugepages-2Mi:          0
 memory:                 7548500Ki
 pods:                   250
-------                      ----                        ------------  ----------  ---------------  -------------  --
-
  default                        cpumanager-6cqz7            1 (66%)       1 (66%)     1G (12%)
1G (12%)      29m

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

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

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

5.2. TOPOLOGY MANAGER POLICIES

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

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

none policy

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

best-effort policy

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

restricted policy

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

single-numa-node policy

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

5.3. SETTING UP TOPOLOGY MANAGER

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

Prerequisites

- Configure the CPU Manager policy to be **static**.

Procedure

To activate Topology Manager:

1. Configure the Topology Manager allocation policy in the custom resource.

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

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

1 This parameter must be **static** with a lowercase **s**.

2 Specify your selected Topology Manager allocation policy. Here, the policy is **single-numa-node**. Acceptable values are: **default**, **best-effort**, **restricted**, **single-numa-node**.

5.4. POD INTERACTIONS WITH TOPOLOGY MANAGER POLICIES

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

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

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

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

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

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

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

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

Topology Manager would consider this pod. The Topology Manager would consult the hint providers, which are CPU Manager and Device Manager, to get topology hints for the pod.

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

CHAPTER 6. 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 in OpenShift Container Platform versions 4.12.0 to 4.12.23 only. It is generally available in OpenShift Container Platform version 4.12.24 and later. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

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.

6.1. ABOUT NUMA-AWARE SCHEDULING

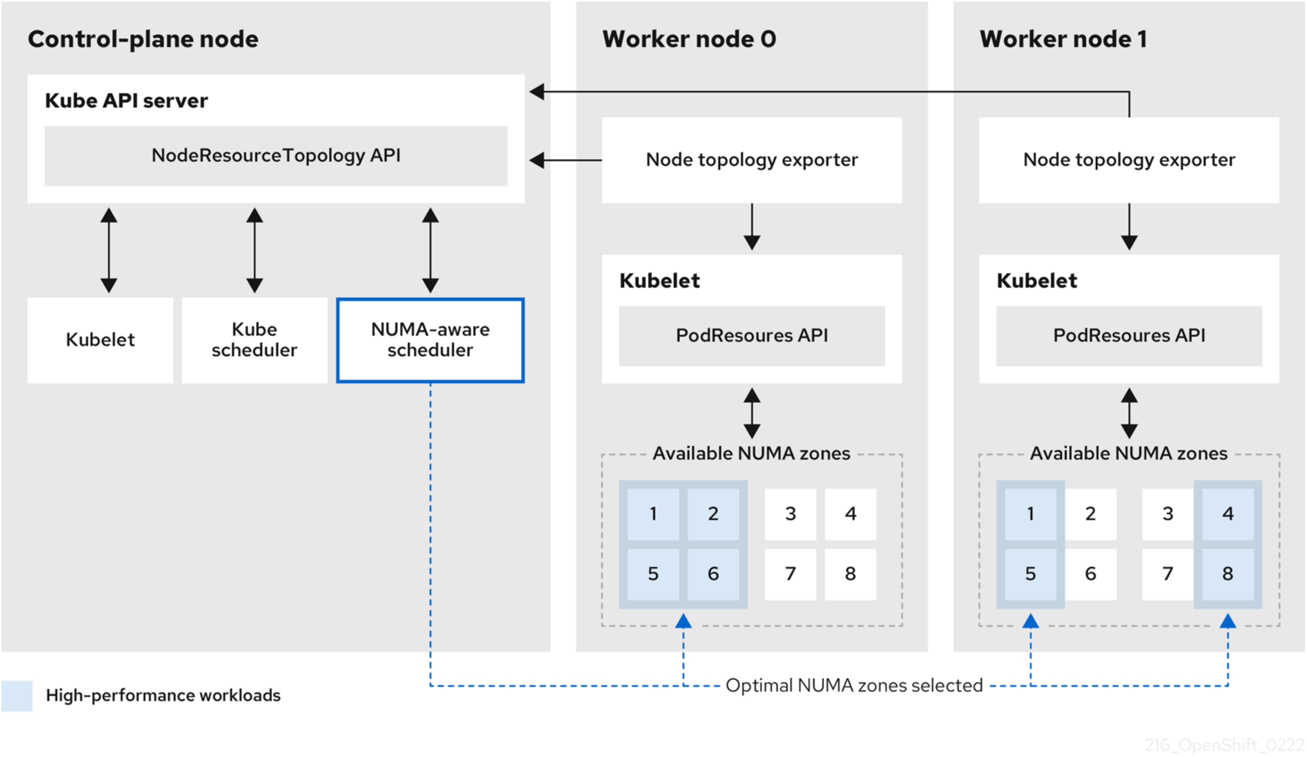
Non-Uniform Memory Access (NUMA) is a compute platform architecture that allows different CPUs to access different regions of memory at different speeds. NUMA resource topology refers to the locations of CPUs, memory, and PCI devices relative to each other in the compute node. Co-located resources are 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 6.1. NUMA-aware scheduling overview



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 .

6.2. 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.

6.2.1. 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: "4.12"
     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.v4.12.2   numaresources-operator   4.12.2            Succeeded
```

6.2.2. 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. Create a namespace for the NUMA Resources Operator:

   a. In the OpenShift Container Platform web console, click **Administration → Namespaces**.

   b. Click **Create Namespace**, enter **openshift-numaresources** in the **Name** field, and then click **Create**.

2. Install the NUMA Resources Operator:

   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**.

   c. In the **Installed Namespaces** field, select the **openshift-numaresources** namespace, and then click **Install**.

3. 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 **openshift-numaresources** namespace with a **Status** of **InstallSucceeded**.

NOTE

During installation an Operator might display a **Failed** status. If the installation later succeeds with an **InstallSucceeded** message, you can ignore the **Failed** message.

If the Operator does not appear as installed, to troubleshoot further:

- Go to the **Operators** → **Installed Operators** page and inspect the **Operator Subscriptions** and **Install Plans** tabs for any failure or errors under **Status**.

- Go to the **Workloads** → **Pods** page and check the logs for pods in the **default** project.

6.3. SCHEDULING NUMA-AWARE WORKLOADS

Clusters running latency-sensitive workloads typically feature performance profiles that help to minimize workload latency and optimize performance. The NUMA-aware scheduler deploys workloads based on available node NUMA resources and with respect to any performance profile settings applied to the node. The combination of NUMA-aware deployments, and the performance profile of the workload, ensures that workloads are scheduled in a way that maximizes performance.

6.3.1. Creating the NUMAResourcesOperator custom resource

When you have installed the NUMA Resources Operator, then create the **NUMAResourcesOperator** custom resource (CR) that instructs the NUMA Resources Operator to install all the cluster infrastructure needed to support the NUMA-aware scheduler, including daemon sets and APIs.

Prerequisites

- Install the OpenShift CLI (**oc**).

- Log in as a user with **cluster-admin** privileges.

- Install the NUMA Resources Operator.

Procedure

1. Create the **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
```

6.3.2. 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:v4.12"
```

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
```

Additional resources

- About the Performance Profile Creator.

6.3.3. 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: ctnr
        image: quay.io/openshifttest/hello-openshift:openshift
        imagePullPolicy: IfNotPresent
        resources:
          limits:
            memory: "100Mi"
            cpu: "10"
          requests:
            memory: "100Mi"
            cpu: "10"
      - name: ctnr2
        image: registry.access.redhat.com/rhel:latest
        imagePullPolicy: IfNotPresent
        command: ["/bin/sh", "-c"]
        args: [ "while true; do sleep 1h; done;" ]
        resources:
          limits:
            memory: "100Mi"
            cpu: "8"
          requests:
            memory: "100Mi"
            cpu: "8"
```

1 **schedulerName** must match the name of the NUMA-aware scheduler that is deployed in your cluster, for example **topo-aware-scheduler**.

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

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

Verification

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
```

6.4. SCHEDULING NUMA-AWARE WORKLOADS WITH MANUAL PERFORMANCE SETTINGS

Clusters running latency-sensitive workloads typically feature performance profiles that help to minimize workload latency and optimize performance. However, you can schedule NUMA-aware workloads in a pristine cluster that does not feature a performance profile. The following workflow features a pristine cluster that you can manually configure for performance by using the **KubeletConfig** resource. This is not the typical environment for scheduling NUMA-aware workloads.

6.4.1. Creating the NUMAResourcesOperator custom resource with manual performance settings

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. Optional: Create the **MachineConfigPool** custom resource that enables custom kubelet configurations for worker nodes:

   NOTE

   By default, OpenShift Container Platform creates a **MachineConfigPool** resource for worker nodes in the cluster. You can create a custom **MachineConfigPool** resource if required.

   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/v1
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
```

6.4.2. Deploying the NUMA-aware secondary pod scheduler with manual performance settings

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"
    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/v1
kind: NUMAResourcesScheduler
metadata:
  name: numaresourcesscheduler
spec:
  imageSpec: "registry.redhat.io/openshift4/noderesourcetopology-scheduler-container-rhel8:v4.12"
  cacheResyncPeriod: "5s" 1
```

1 Enter an interval value in seconds for synchronization of the scheduler cache. A value of **5s** is typical for most implementations.

NOTE

- Enable the **cacheResyncPeriod** specification to help the NUMA Resource Operator report more exact resource availability by monitoring pending resources on nodes and synchronizing this information in the scheduler cache at a defined interval. This also helps to minimize **Topology Affinity Error** errors because of sub-optimal scheduling decisions. The lower the interval the greater the network load. The **cacheResyncPeriod** specification is disabled by default.

- Setting a value of **Enabled** for the **podsFingerprinting** specification in the **NUMAResourcesOperator** CR is a requirement for the implementation of the **cacheResyncPeriod** specification.

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
```

6.4.3. Scheduling workloads with the NUMA-aware scheduler with manual performance settings

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: ctnr
           image: quay.io/openshifttest/hello-openshift:openshift
           imagePullPolicy: IfNotPresent
           resources:
             limits:
               memory: "100Mi"
               cpu: "10"
             requests:
               memory: "100Mi"
               cpu: "10"
         - name: ctnr2
           image: registry.access.redhat.com/rhel:latest
           imagePullPolicy: IfNotPresent
           command: ["/bin/sh", "-c"]
           args: [ "while true; do sleep 1h; done;" ]
           resources:
             limits:
               memory: "100Mi"
   ```

```
          cpu: "8"
        requests:
          memory: "100Mi"
          cpu: "8"
```

1 **schedulerName** must match the name of the NUMA-aware scheduler that is deployed in your cluster, for example **topo-aware-scheduler**.

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

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

Verification

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 are available for scheduling will fail with a **MinimumReplicasUnavailable** error. The deployment succeeds when the required resources become available. Pods remain in the **Pending** state until the required resources are available.

3. Verify that the expected allocated resources are listed for the node.

a. Identify the node that is running the deployment pod by running the following command, replacing <namespace> with the namespace you specified in the **Deployment** CR:

```
$ oc get pods -n <namespace> -o wide
```

Example output

```
NAME                              READY   STATUS    RESTARTS   AGE   IP            NODE
NOMINATED NODE   READINESS GATES
numa-deployment-1-65684f8fcc-bw4bw   0/2     Running   0          82m   10.128.2.50
worker-0   <none>  <none>
```

b. Run the following command, replacing <node_name> with the name of that node that is running the deployment pod:

```
$ oc describe noderesourcetopologies.topology.node.k8s.io <node_name>
```

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** and the CPU request is an integer value, not a decimal value. You can verify the that the pod has a **qosClass** of **Guaranteed** by running the following command:

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

Example output

```
Guaranteed
```

6.5. OPTIONAL: CONFIGURING POLLING OPERATIONS FOR NUMA RESOURCES UPDATES

The daemons controlled by the NUMA Resources Operator in their **nodeGroup** poll resources to retrieve updates about available NUMA resources. You can fine-tune polling operations for these daemons by configuring the **spec.nodeGroups** specification in the **NUMAResourcesOperator** custom resource (CR). This provides advanced control of polling operations. Configure these specifications to improve scheduling behaviour and troubleshoot suboptimal scheduling decisions.

The configuration options are the following:

- **infoRefreshMode**: Determines the trigger condition for polling the kubelet. The NUMA Resources Operator reports the resulting information to the API server.
- **infoRefreshPeriod**: Determines the duration between polling updates.
- **podsFingerprinting**: Determines if point-in-time information for the current set of pods running on a node is exposed in polling updates.

NOTE

podsFingerprinting is enabled by default. **podsFingerprinting** is a requirement for the **cacheResyncPeriod** specification in the **NUMAResourcesScheduler** CR. The **cacheResyncPeriod** specification helps to report more exact resource availability by monitoring pending resources on nodes.

Prerequisites

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

Procedure

- Configure the **spec.nodeGroups** specification in your **NUMAResourcesOperator** CR:

```
apiVersion: nodetopology.openshift.io/v1
kind: NUMAResourcesOperator
metadata:
  name: numaresourcesoperator
spec:
  nodeGroups:
```

```
- config:
    infoRefreshMode: Periodic 1
    infoRefreshPeriod: 10s 2
    podsFingerprinting: Enabled 3
  name: worker
```

1. Valid values are **Periodic**, **Events**, **PeriodicAndEvents**. Use **Periodic** to poll the kubelet at intervals that you define in **infoRefreshPeriod**. Use **Events** to poll the kubelet at every pod lifecycle event. Use **PeriodicAndEvents** to enable both methods.

2. Define the polling interval for **Periodic** or **PeriodicAndEvents** refresh modes. The field is ignored if the refresh mode is **Events**.

3. Valid values are **Enabled** or **Disabled**. Setting to **Enabled** is a requirement for the **cacheResyncPeriod** specification in the **NUMAResourcesScheduler**.

Verification

1. After you deploy the NUMA Resources Operator, verify that the node group configurations were applied by running the following command:

```
$ oc get numaresop numaresourcesoperator -o json | jq '.status'
```

Example output

```
...

    "config": {
    "infoRefreshMode": "Periodic",
    "infoRefreshPeriod": "10s",
    "podsFingerprinting": "Enabled"
  },
  "name": "worker"

...
```

6.6. 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:
       k8stopoawareschedwg/rte-update: periodic
```

```
  creationTimestamp: "2022-06-16T08:55:37Z"
  generation: 62061
  name: worker-1
  resourceVersion: "8450129"
  uid: e8659390-6f8d-4e67-9a51-1ea34bba1cc3
topologyPolicies:
- SingleNUMANodeContainerLevel
zones: 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.

6.6.1. 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:v4.12"
  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 succesfully:

   ```
   $ 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"
```

6.6.2. 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
```

4. Examine the logs of the **resource-topology-exporter** container running on the worker pod that corresponds to the node you are troubleshooting. Run the following command:

```
$ oc logs -n openshift-numaresources -c resource-topology-exporter numaresourcesoperator-
worker-pb75c
```

Example output

```
I0221 13:38:18.334140       1 main.go:206] using sysinfo:
reservedCpus: 0,1
reservedMemory:
  "0": 1178599424
I0221 13:38:18.334370       1 main.go:67] === System information ===
I0221 13:38:18.334381       1 sysinfo.go:231] cpus: reserved "0-1"
I0221 13:38:18.334493       1 sysinfo.go:237] cpus: online "0-103"
I0221 13:38:18.546750       1 main.go:72]
cpus: allocatable "2-103"
hugepages-1Gi:
  numa cell 0 -> 6
  numa cell 1 -> 1
hugepages-2Mi:
  numa cell 0 -> 64
  numa cell 1 -> 128
memory:
  numa cell 0 -> 45758Mi
  numa cell 1 -> 48372Mi
```

6.6.3. 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 7. SCALABILITY AND PERFORMANCE OPTIMIZATION

7.1. OPTIMIZING STORAGE

Optimizing storage helps to minimize storage use across all resources. By optimizing storage, administrators help ensure that existing storage resources are working in an efficient manner.

7.1.1. Available persistent storage options

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

Table 7.1. Available storage options

Storage type	Description	Examples
Block	● 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	● 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	● Accessible through a REST API endpoint ● Configurable for use in the OpenShift image 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 plugin.

7.1.2. Recommended configurable storage technology

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

Table 7.2. Recommended and configurable storage technology

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

[1] **ReadOnlyMany**

[2] **ReadWriteMany**

[3] Prometheus is the underlying technology used for metrics.

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

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

[6] For logging, review the recommended storage solution in Configuring persistent storage for the log store section. Using NFS storage as a persistent volume or through NAS, such as Gluster, can corrupt the data. Hence, NFS is not supported for Elasticsearch storage and LokiStack log store in OpenShift Container Platform Logging. You must use one persistent volume type per log store.

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

NOTE

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

7.1.2.1. 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.

7.1.2.1.1. Registry

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

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

7.1.2.1.2. Scaled registry

In a scaled/HA OpenShift image registry cluster deployment:

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

7.1.2.1.3. Metrics

In an OpenShift Container Platform hosted metrics cluster deployment:

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

IMPORTANT

It is not recommended to use file storage for a hosted metrics cluster deployment with production workloads.

7.1.2.1.4. Logging

In an OpenShift Container Platform hosted logging cluster deployment:

- Loki Operator:
  - The preferred storage technology is S3 compatible Object storage.
  - Block storage is not configurable.
- OpenShift Elasticsearch Operator:
  - The preferred storage technology is block storage.
  - Object storage is not supported.

NOTE

As of logging version 5.4.3 the OpenShift Elasticsearch Operator is deprecated and is planned to be removed in a future release. Red Hat will provide bug fixes and support for this feature during the current release lifecycle, but this feature will no longer receive enhancements and will be removed. As an alternative to using the OpenShift Elasticsearch Operator to manage the default log storage, you can use the Loki Operator.

7.1.2.1.5. Applications

Application use cases vary from application to application, as described in the following examples:

- Storage technologies that support dynamic PV provisioning have low mount time latencies, and are not tied to nodes to support a healthy cluster.
- Application developers are responsible for knowing and understanding the storage requirements for their application, and how it works with the provided storage to ensure that issues do not occur when an application scales or interacts with the storage layer.

7.1.2.2. Other specific application storage recommendations

IMPORTANT

It is not recommended to use RAID configurations on **Write** intensive workloads, such as **etcd**. If you are running **etcd** with a RAID configuration, you might be at risk of encountering performance issues with your workloads.

- Red Hat OpenStack Platform (RHOSP) Cinder: RHOSP Cinder tends to be adept in ROX access mode use cases.

- Databases: Databases (RDBMSs, NoSQL DBs, etc.) tend to perform best with dedicated block storage.
- The etcd database must have enough storage and adequate performance capacity to enable a large cluster. Information about monitoring and benchmarking tools to establish ample storage and a high-performance environment is described in *Recommended etcd practices*.

7.1.3. Data storage management

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

Table 7.3. Main directories for storing OpenShift Container Platform data

Directory	Notes	Sizing	Expected growth
/var/lib/etcd	Used for etcd storage when storing the database.	Less than 20 GB. Database can grow up to 8 GB.	Will grow slowly with the environment. Only storing metadata. Additional 20-25 GB for every additional 8 GB of memory.
/var/lib/containers	This is the mount point for the CRI-O runtime. Storage used for active container runtimes, including pods, and storage of local images. Not used for registry storage.	50 GB for a node with 16 GB memory. Note that this sizing should not be used to determine minimum cluster requirements. Additional 20-25 GB for every additional 8 GB of memory.	Growth is limited by capacity for running containers.
/var/lib/kubelet	Ephemeral volume storage for pods. This includes anything external that is mounted into a container at runtime. Includes environment variables, kube secrets, and data volumes not backed by persistent volumes.	Varies	Minimal if pods requiring storage are using persistent volumes. If using ephemeral storage, this can grow quickly.
/var/log	**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.**

7.1.4. Optimizing storage performance for Microsoft Azure

OpenShift Container Platform and Kubernetes are sensitive to disk performance, and faster storage is recommended, particularly for etcd on the control plane nodes.

For production Azure clusters and clusters with intensive workloads, the virtual machine operating system disk for control plane machines should be able to sustain a tested and recommended minimum throughput of 5000 IOPS / 200MBps. This throughput can be provided by having a minimum of 1 TiB Premium SSD (P30). In Azure and Azure Stack Hub, disk performance is directly dependent on SSD disk sizes. To achieve the throughput supported by a **Standard_D8s_v3** virtual machine, or other similar machine types, and the target of 5000 IOPS, at least a P30 disk is required.

Host caching must be set to **ReadOnly** for low latency and high IOPS and throughput when reading data. Reading data from the cache, which is present either in the VM memory or in the local SSD disk, is much faster than reading from the disk, which is in the blob storage.

7.1.5. Additional resources

- Configuring the Elasticsearch log store

7.2. OPTIMIZING ROUTING

The OpenShift Container Platform HAProxy router can be scaled or configured to optimize performance.

7.2.1. Baseline Ingress Controller (router) performance

The OpenShift Container Platform Ingress Controller, or router, is the ingress point for ingress traffic for applications and services that are configured using routes and ingresses.

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

- HTTP keep-alive/close mode
- Route type
- TLS session resumption client support
- Number of concurrent connections per target route
- Number of target routes
- Back end server page size
- Underlying infrastructure (network/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
re-encrypt	2320	2941

The default Ingress Controller configuration was used with the **spec.tuningOptions.threadCount** field set to **4**. Two different endpoint publishing strategies were tested: Load Balancer Service and Host Network. TLS session resumption was used for encrypted routes. With HTTP keep-alive, a single HAProxy router is capable of saturating a 1 Gbit NIC at page sizes as small as 8 kB.

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

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

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

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

For more information on Ingress sharding, see Configuring Ingress Controller sharding by using route labels and Configuring Ingress Controller sharding by using namespace labels .

You can modify the Ingress Controller deployment using the information provided in Setting Ingress Controller thread count for threads and Ingress Controller configuration parameters for timeouts, and other tuning configurations in the Ingress Controller specification.

7.2.2. Configuring Ingress Controller liveness, readiness, and startup probes

Cluster administrators can configure the timeout values for the kubelet's liveness, readiness, and startup probes for router deployments that are managed by the OpenShift Container Platform Ingress Controller (router). The liveness and readiness probes of the router use the default timeout value of 1 second, which is too brief when networking or runtime performance is severely degraded. Probe timeouts can cause unwanted router restarts that interrupt application connections. The ability to set larger timeout values can reduce the risk of unnecessary and unwanted restarts.

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

Parameter	Description
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
```

7.2.3. Configuring HAProxy reload interval

When you update a route or an endpoint associated with a route, OpenShift Container Platform router updates the configuration for HAProxy. Then, HAProxy reloads the updated configuration for those changes to take effect. When HAProxy reloads, it generates a new process that handles new connections using the updated configuration.

HAProxy keeps the old process running to handle existing connections until those connections are all closed. When old processes have long-lived connections, these processes can accumulate and consume resources.

The default minimum HAProxy reload interval is five seconds. You can configure an Ingress Controller using its **spec.tuningOptions.reloadInterval** field to set a longer minimum reload interval.

> **WARNING**
>
> Setting a large value for the minimum HAProxy reload interval can cause latency in observing updates to routes and their endpoints. To lessen the risk, avoid setting a value larger than the tolerable latency for updates.

Procedure

- Change the minimum HAProxy reload interval of the default Ingress Controller to 15 seconds by running the following command:

```
$ oc -n openshift-ingress-operator patch ingresscontrollers/default --type=merge --
patch='{"spec":{"tuningOptions":{"reloadInterval":"15s"}}}'
```

7.3. 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, multi-queue, and ethtool settings.

OVN-Kubernetes uses Generic Network Virtualization Encapsulation (Geneve) instead of VXLAN as the tunnel protocol. This network can be tuned by using network interface controller (NIC) offloads.

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 plugins 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.

7.3.1. Optimizing the MTU for your network

There are two important maximum transmission units (MTUs): the network interface controller (NIC) MTU and the cluster network MTU.

The NIC MTU is 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 OpenShift SDN network plugin overlay 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, this should be set to **1450**. On a jumbo frame ethernet network, this should be set to **8950**. These values should be set automatically by the Cluster Network Operator based on the NIC’s configured MTU. Therefore, cluster administrators do not typically update these values. Amazon Web Services (AWS) and bare-metal environments support jumbo frame ethernet networks. This setting will help throughput, especially with transmission control protocol (TCP).

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 network plugin. Other SDN solutions might require the value to be more or less.

7.3.2. Recommended practices for installing large scale clusters

When installing large clusters or scaling the cluster to larger node counts, set the cluster network **cidr** accordingly in your **install-config.yaml** file before you install the cluster:

```
networking:
  clusterNetwork:
  - cidr: 10.128.0.0/14
    hostPrefix: 23
  machineNetwork:
  - cidr: 10.0.0.0/16
  networkType: OVNKubernetes
  serviceNetwork:
  - 172.30.0.0/16
```

The default cluster network **cidr 10.128.0.0/14** cannot be used if the cluster size is more than 500 nodes. 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.

7.3.3. Impact of IPsec

Because encrypting and decrypting node hosts uses CPU power, performance is affected both in throughput and CPU usage on the nodes when encryption is enabled, regardless of the IP security system being used.

IPSec encrypts traffic at the IP payload level, before it hits the NIC, protecting fields that would otherwise be used for NIC offloading. This means that some NIC acceleration features might not be usable when IPSec is enabled and will lead to decreased throughput and increased CPU usage.

7.3.4. Additional resources

- Modifying advanced network configuration parameters
- Configuration parameters for the OVN-Kubernetes network plugin
- Configuration parameters for the OpenShift SDN network plugin
- Improving cluster stability in high latency environments using worker latency profiles

7.4. OPTIMIZING CPU USAGE WITH MOUNT NAMESPACE ENCAPSULATION

You can optimize CPU usage in OpenShift Container Platform clusters by using mount namespace encapsulation to provide a private namespace for kubelet and CRI-O processes. This reduces the cluster CPU resources used by systemd with no difference in functionality.

IMPORTANT

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

For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

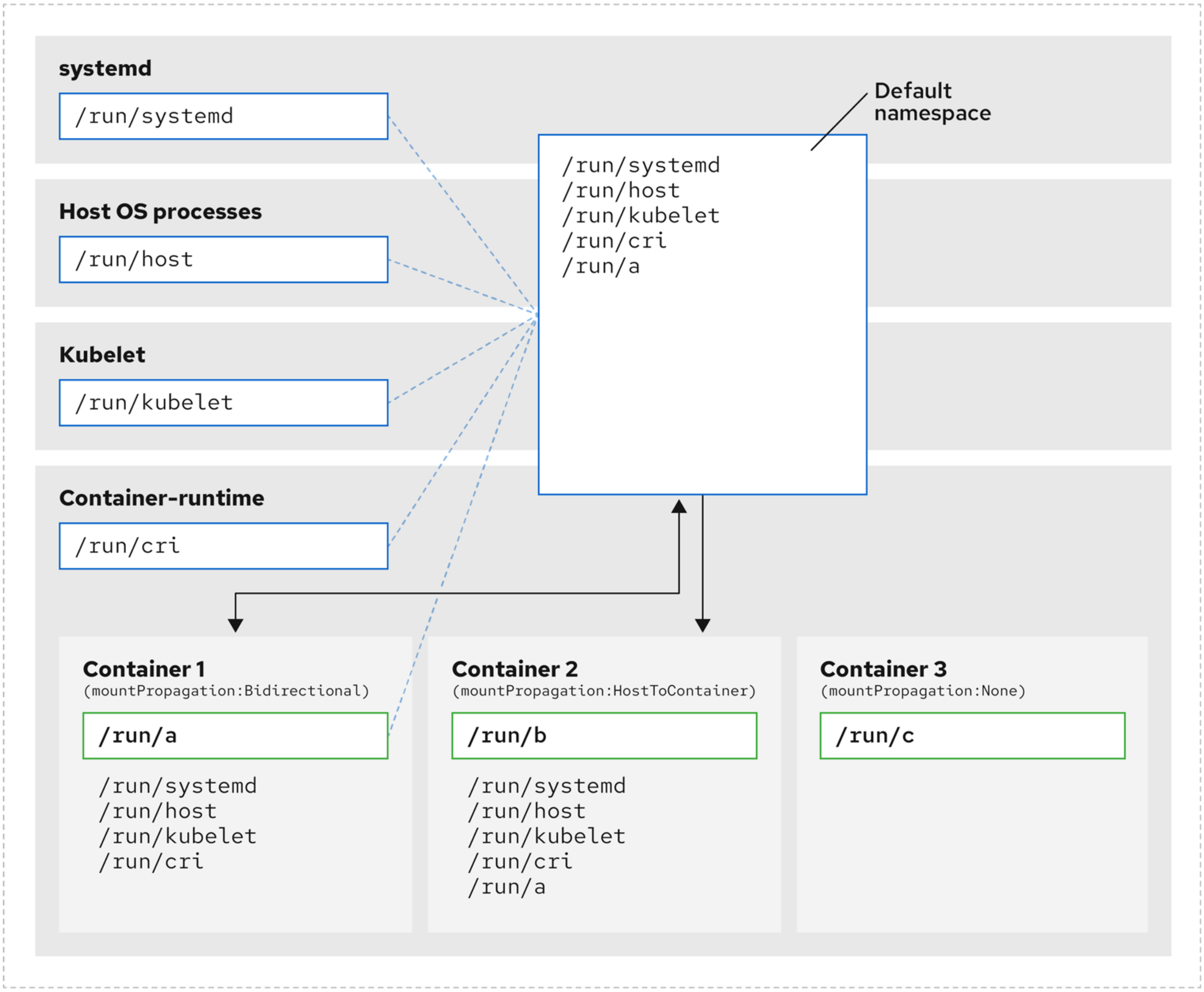
7.4.1. Encapsulating mount namespaces

Mount namespaces are used to isolate mount points so that processes in different namespaces cannot view each others' files. Encapsulation is the process of moving Kubernetes mount namespaces to an alternative location where they will not be constantly scanned by the host operating system.

The host operating system uses systemd to constantly scan all mount namespaces: both the standard Linux mounts and the numerous mounts that Kubernetes uses to operate. The current implementation of kubelet and CRI-O both use the top-level namespace for all container runtime and kubelet mount points. However, encapsulating these container-specific mount points in a private namespace reduces systemd overhead with no difference in functionality. Using a separate mount namespace for both CRI-O and kubelet can encapsulate container-specific mounts from any systemd or other host operating system interaction.

This ability to potentially achieve major CPU optimization is now available to all OpenShift Container Platform administrators. Encapsulation can also improve security by storing Kubernetes-specific mount points in a location safe from inspection by unprivileged users.

The following diagrams illustrate a Kubernetes installation before and after encapsulation. Both scenarios show example containers which have mount propagation settings of bidirectional, host-to-container, and none.



290_OpenShift_1122

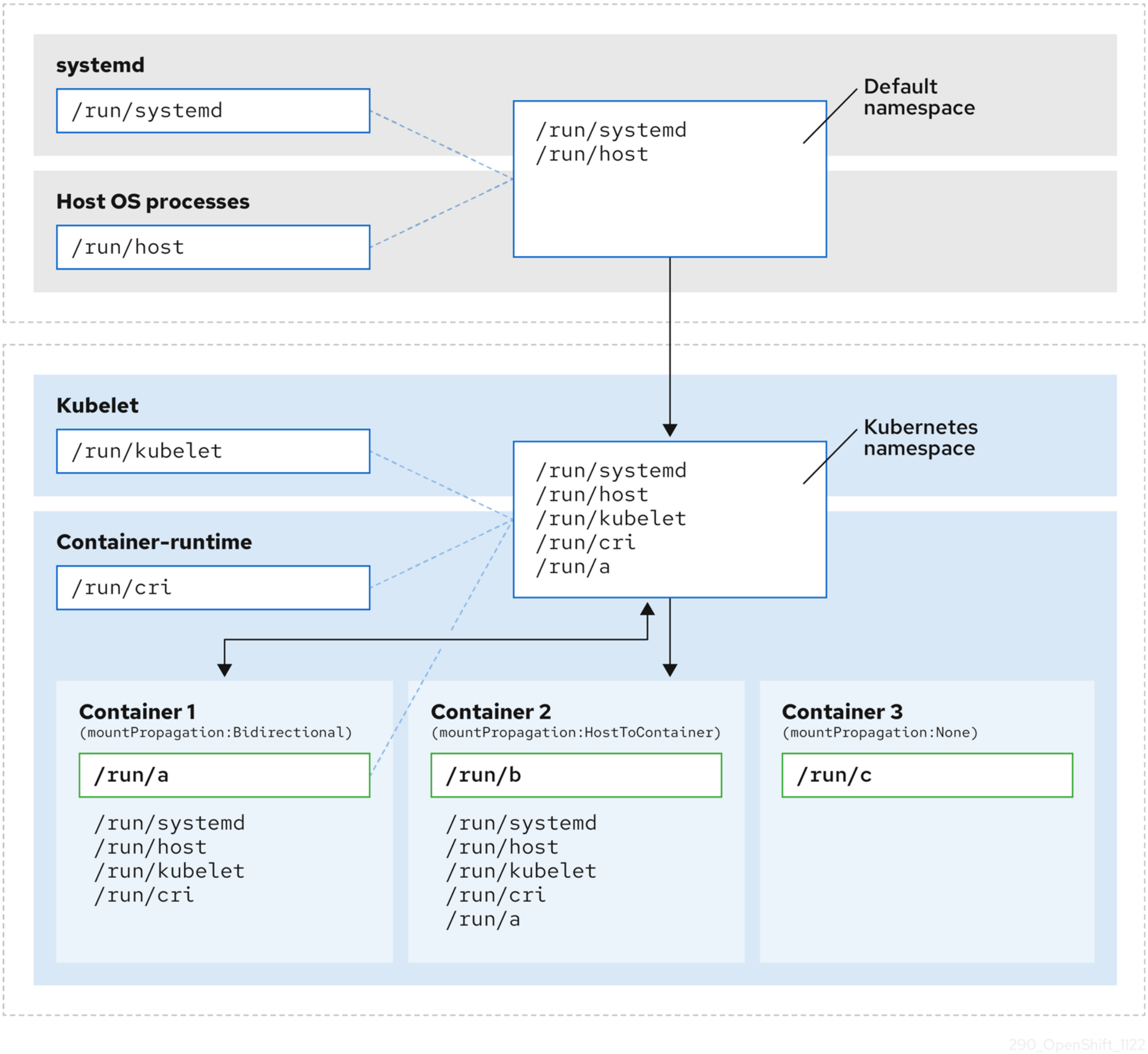
Here we see systemd, host operating system processes, kubelet, and the container runtime sharing a single mount namespace.

- systemd, host operating system processes, kubelet, and the container runtime each have access to and visibility of all mount points.
- Container 1, configured with bidirectional mount propagation, can access systemd and host mounts, kubelet and CRI-O mounts. A mount originating in Container 1, such as **/run/a** is visible to systemd, host operating system processes, kubelet, container runtime, and other containers with host-to-container or bidirectional mount propagation configured (as in Container 2).

- Container 2, configured with host-to-container mount propagation, can access systemd and host mounts, kubelet and CRI-O mounts. A mount originating in Container 2, such as **/run/b**, is not visible to any other context.

- Container 3, configured with no mount propagation, has no visibility of external mount points. A mount originating in Container 3, such as **/run/c**, is not visible to any other context.

The following diagram illustrates the system state after encapsulation.



- The main systemd process is no longer devoted to unnecessary scanning of Kubernetes-specific mount points. It only monitors systemd-specific and host mount points.

- The host operating system processes can access only the systemd and host mount points.

- Using a separate mount namespace for both CRI-O and kubelet completely separates all container-specific mounts away from any systemd or other host operating system interaction whatsoever.

- The behavior of Container 1 is unchanged, except a mount it creates such as **/run/a** is no longer visible to systemd or host operating system processes. It is still visible to kubelet, CRI-O, and other containers with host-to-container or bidirectional mount propagation configured (like Container 2).

- The behavior of Container 2 and Container 3 is unchanged.

7.4.2. Configuring mount namespace encapsulation

You can configure mount namespace encapsulation so that a cluster runs with less resource overhead.

> **NOTE**
>
> Mount namespace encapsulation is a Technology Preview feature and it is disabled by default. To use it, you must enable the feature manually.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have logged in as a user with **cluster-admin** privileges.

Procedure

1. Create a file called **mount_namespace_config.yaml** with the following YAML:

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: 99-kubens-master
spec:
  config:
    ignition:
      version: 3.2.0
    systemd:
      units:
      - enabled: true
        name: kubens.service
---
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: 99-kubens-worker
spec:
  config:
    ignition:
      version: 3.2.0
    systemd:
      units:
      - enabled: true
        name: kubens.service
```

2. Apply the mount namespace **MachineConfig** CR by running the following command:

```
$ oc apply -f mount_namespace_config.yaml
```

Example output

```
machineconfig.machineconfiguration.openshift.io/99-kubens-master created
machineconfig.machineconfiguration.openshift.io/99-kubens-worker created
```

3. The **MachineConfig** CR can take up to 30 minutes to finish being applied in the cluster. You can check the status of the **MachineConfig** CR by running the following command:

```
$ oc get mcp
```

Example output

```
NAME     CONFIG                                     UPDATED   UPDATING   DEGRADED
MACHINECOUNT   READYMACHINECOUNT   UPDATEDMACHINECOUNT
DEGRADEDMACHINECOUNT   AGE
master   rendered-master-03d4bc4befb0f4ed3566a2c8f7636751   False     True       False
3           0                 0                   0                    45m
worker   rendered-worker-10577f6ab0117ed1825f8af2ac687ddf   False     True       False
3           1                 1
```

4. Wait for the **MachineConfig** CR to be applied successfully across all control plane and worker nodes after running the following command:

```
$ oc wait --for=condition=Updated mcp --all --timeout=30m
```

Example output

```
machineconfigpool.machineconfiguration.openshift.io/master condition met
machineconfigpool.machineconfiguration.openshift.io/worker condition met
```

Verification

To verify encapsulation for a cluster host, run the following commands:

1. Open a debug shell to the cluster host:

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

2. Open a **chroot** session:

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

3. Check the systemd mount namespace:

```
sh-4.4# readlink /proc/1/ns/mnt
```

Example output

```
mnt:[4026531953]
```

4. Check kubelet mount namespace:

```
sh-4.4# readlink /proc/$(pgrep kubelet)/ns/mnt
```

Example output

```
mnt:[4026531840]
```

5. Check the CRI-O mount namespace:

```
sh-4.4# readlink /proc/$(pgrep crio)/ns/mnt
```

Example output

```
mnt:[4026531840]
```

These commands return the mount namespaces associated with systemd, kubelet, and the container runtime. In OpenShift Container Platform, the container runtime is CRI-O.

Encapsulation is in effect if systemd is in a different mount namespace to kubelet and CRI-O as in the above example. Encapsulation is not in effect if all three processes are in the same mount namespace.

7.4.3. Inspecting encapsulated namespaces

You can inspect Kubernetes-specific mount points in the cluster host operating system for debugging or auditing purposes by using the **kubensenter** script that is available in Red Hat Enterprise Linux CoreOS (RHCOS).

SSH shell sessions to the cluster host are in the default namespace. To inspect Kubernetes-specific mount points in an SSH shell prompt, you need to run the **kubensenter** script as root. The **kubensenter** script is aware of the state of the mount encapsulation, and is safe to run even if encapsulation is not enabled.

NOTE

oc debug remote shell sessions start inside the Kubernetes namespace by default. You do not need to run **kubensenter** to inspect mount points when you use **oc debug**.

If the encapsulation feature is not enabled, the **kubensenter findmnt** and **findmnt** commands return the same output, regardless of whether they are run in an **oc debug** session or in an SSH shell prompt.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have logged in as a user with **cluster-admin** privileges.
- You have configured SSH access to the cluster host.

Procedure

1. Open a remote SSH shell to the cluster host. For example:

```
$ ssh core@<node_name>
```

2. Run commands using the provided **kubensenter** script as the root user. To run a single command inside the Kubernetes namespace, provide the command and any arguments to the **kubensenter** script. For example, to run the **findmnt** command inside the Kubernetes namespace, run the following command:

```
[core@control-plane-1 ~]$ sudo kubensenter findmnt
```

Example output

```
kubensenter: Autodetect: kubens.service namespace found at /run/kubens/mnt
TARGET                              SOURCE              FSTYPE    OPTIONS
/
/dev/sda4[/ostree/deploy/rhcos/deploy/32074f0e8e5ec453e56f5a8a7bc9347eaa4172349ceab9
c22b709d9d71a3f4b0.0]
|                                               xfs
rw,relatime,seclabel,attr2,inode64,logbufs=8,logbsize=32k,prjquota
                             shm                 tmpfs
...
```

3. To start a new interactive shell inside the Kubernetes namespace, run the **kubensenter** script without any arguments:

```
[core@control-plane-1 ~]$ sudo kubensenter
```

Example output

```
kubensenter: Autodetect: kubens.service namespace found at /run/kubens/mnt
```

7.4.4. Running additional services in the encapsulated namespace

Any monitoring tool that relies on the ability to run in the host operating system and have visibility of mount points created by kubelet, CRI-O, or containers themselves, must enter the container mount namespace to see these mount points. The **kubensenter** script that is provided with OpenShift Container Platform executes another command inside the Kubernetes mount point and can be used to adapt any existing tools.

The **kubensenter** script is aware of the state of the mount encapsulation feature status, and is safe to run even if encapsulation is not enabled. In that case the script executes the provided command in the default mount namespace.

For example, if a systemd service needs to run inside the new Kubernetes mount namespace, edit the service file and use the **ExecStart=** command line with **kubensenter**.

```
[Unit]
Description=Example service
[Service]
ExecStart=/usr/bin/kubensenter /path/to/original/command arg1 arg2
```

7.4.5. Additional resources

- What are namespaces
- Manage containers in namespaces by using nsenter
- MachineConfig

CHAPTER 8. 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.

8.1. ABOUT BARE METAL HOSTS AND NODES

To provision a Red Hat Enterprise Linux CoreOS (RHCOS) bare metal host as a node in your cluster, first create a **MachineSet** custom resource (CR) object that corresponds to the bare metal host hardware. Bare metal host compute machine sets describe infrastructure components specific to your configuration. You apply specific Kubernetes labels to these compute machine sets and then update the infrastructure components to run on only those machines.

Machine CR's are created automatically when you scale up the relevant **MachineSet** containing a **metal3.io/autoscale-to-hosts** annotation. OpenShift Container Platform uses **Machine** CR's to provision the bare metal node that corresponds to the host as specified in the **MachineSet** CR.

8.2. MAINTAINING BARE METAL HOSTS

You can maintain the details of the bare metal hosts in your cluster from the OpenShift Container Platform web console. Navigate to **Compute → Bare Metal Hosts**, and select a task from the **Actions** drop down menu. Here you can manage items such as BMC details, boot MAC address for the host, enable power management, and so on. You can also review the details of the network interfaces and drives for the host.

You can move a bare metal host into maintenance mode. When you move a host into maintenance mode, the scheduler moves all managed workloads off the corresponding bare metal node. No new workloads are scheduled while in maintenance mode.

You can deprovision a bare metal host in the web console. Deprovisioning a host does the following actions:

1. Annotates the bare metal host CR with **cluster.k8s.io/delete-machine: true**

2. Scales down the related compute machine set

NOTE

Powering off the host without first moving the daemon set and unmanaged static pods to another node can cause service disruption and loss of data.

Additional resources

- Adding compute machines to bare metal

8.2.1. Adding a bare metal host to the cluster using the web console

You can add bare metal hosts to the cluster in the web console.

Prerequisites

- Install an RHCOS cluster on bare metal.

- Log in as a user with **cluster-admin** privileges.

Procedure

1. In the web console, navigate to **Compute → Bare Metal Hosts**.
2. Select **Add Host → New with Dialog**.
3. Specify a unique name for the new bare metal host.
4. Set the **Boot MAC address**.
5. Set the **Baseboard Management Console (BMC) Address**
6. Enter the user credentials for the host's baseboard management controller (BMC).
7. Select to power on the host after creation, and select **Create**.
8. Scale up the number of replicas to match the number of available bare metal hosts. Navigate to **Compute → MachineSets**, and increase the number of machine replicas in the cluster by selecting **Edit Machine count** from the **Actions** drop-down menu.

NOTE

You can also manage the number of bare metal nodes using the **oc scale** command and the appropriate bare metal compute machine set.

8.2.2. Adding a bare metal host to the cluster using YAML in the web console

You can add bare metal hosts to the cluster in the web console using a YAML file that describes the bare metal host.

Prerequisites

- Install a RHCOS compute machine on bare metal infrastructure for use in the cluster.
- Log in as a user with **cluster-admin** privileges.
- Create a **Secret** CR for the bare metal host.

Procedure

1. In the web console, navigate to **Compute → Bare Metal Hosts**.
2. Select **Add Host → New from YAML**.
3. Copy and paste the below YAML, modifying the relevant fields with the details of your host:

```
apiVersion: metal3.io/v1alpha1
kind: BareMetalHost
metadata:
  name: <bare_metal_host_name>
spec:
  online: true
  bmc:
    address: <bmc_address>
    credentialsName: <secret_credentials_name> 1
    disableCertificateVerification: True 2
  bootMACAddress: <host_boot_mac_address>
```

1 **credentialsName** must reference a valid **Secret** CR. The **baremetal-operator** cannot manage the bare metal host without a valid **Secret** referenced in the **credentialsName**. For more information about secrets and how to create them, see Understanding secrets.

2 Setting **disableCertificateVerification** to **true** disables TLS host validation between the cluster and the baseboard management controller (BMC).

4. Select **Create** to save the YAML and create the new bare metal host.

5. Scale up the number of replicas to match the number of available bare metal hosts. Navigate to **Compute** → **MachineSets**, and increase the number of machines in the cluster by selecting **Edit Machine count** from the **Actions** drop-down menu.

> NOTE
>
> You can also manage the number of bare metal nodes using the **oc scale** command and the appropriate bare metal compute machine set.

8.2.3. Automatically scaling machines to the number of available bare metal hosts

To automatically create the number of **Machine** objects that matches the number of available **BareMetalHost** objects, add a **metal3.io/autoscale-to-hosts** annotation to the **MachineSet** object.

Prerequisites

- Install RHCOS bare metal compute machines for use in the cluster, and create corresponding **BareMetalHost** objects.
- Install the OpenShift Container Platform CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

Procedure

1. Annotate the compute machine set that you want to configure for automatic scaling by adding the **metal3.io/autoscale-to-hosts** annotation. Replace **<machineset>** with the name of the compute machine set.

   ```
   $ oc annotate machineset <machineset> -n openshift-machine-api 'metal3.io/autoscale-to-hosts=<any_value>'
   ```

   Wait for the new scaled machines to start.

NOTE

When you use a **BareMetalHost** object to create a machine in the cluster and labels or selectors are subsequently changed on the **BareMetalHost**, the **BareMetalHost** object continues be counted against the **MachineSet** that the **Machine** object was created from.

8.2.4. Removing bare metal hosts from the provisioner node

In certain circumstances, you might want to temporarily remove bare metal hosts from the provisioner node. For example, during provisioning when a bare metal host reboot is triggered by using the OpenShift Container Platform administration console or as a result of a Machine Config Pool update, OpenShift Container Platform logs into the integrated Dell Remote Access Controller (iDrac) and issues a delete of the job queue.

To prevent the management of the number of **Machine** objects that matches the number of available **BareMetalHost** objects, add a **baremetalhost.metal3.io/detached** annotation to the **MachineSet** object.

NOTE

This annotation has an effect for only **BareMetalHost** objects that are in either **Provisioned**, **ExternallyProvisioned** or **Ready**/**Available** state.

Prerequisites

- Install RHCOS bare metal compute machines for use in the cluster and create corresponding **BareMetalHost** objects.
- Install the OpenShift Container Platform CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

Procedure

1. Annotate the compute machine set that you want to remove from the provisioner node by adding the **baremetalhost.metal3.io/detached** annotation.

```
$ oc annotate machineset <machineset> -n openshift-machine-api
'baremetalhost.metal3.io/detached'
```

   Wait for the new machines to start.

   NOTE

   When you use a **BareMetalHost** object to create a machine in the cluster and labels or selectors are subsequently changed on the **BareMetalHost**, the **BareMetalHost** object continues be counted against the **MachineSet** that the **Machine** object was created from.

2. In the provisioning use case, remove the annotation after the reboot is complete by using the following command:

```
$ oc annotate machineset <machineset> -n openshift-machine-api
'baremetalhost.metal3.io/detached-'
```

Additional resources

- Expanding the cluster
- MachineHealthChecks on bare metal

CHAPTER 9. MONITORING BARE-METAL EVENTS WITH THE BARE METAL EVENT RELAY

IMPORTANT

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

For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

9.1. ABOUT BARE-METAL EVENTS

Use the Bare Metal Event Relay to subscribe applications that run in your OpenShift Container Platform cluster to events that are generated on the underlying bare-metal host. The Redfish service publishes events on a node and transmits them on an advanced message queue to subscribed applications.

Bare-metal events are based on the open Redfish standard that is developed under the guidance of the Distributed Management Task Force (DMTF). Redfish provides a secure industry-standard protocol with a REST API. The protocol is used for the management of distributed, converged or software-defined resources and infrastructure.

Hardware-related events published through Redfish includes:

- Breaches of temperature limits
- Server status
- Fan status

Begin using bare-metal events by deploying the Bare Metal Event Relay Operator and subscribing your application to the service. The Bare Metal Event Relay Operator installs and manages the lifecycle of the Redfish bare-metal event service.

NOTE

The Bare Metal Event Relay works only with Redfish-capable devices on single-node clusters provisioned on bare-metal infrastructure.

9.2. HOW BARE-METAL EVENTS WORK

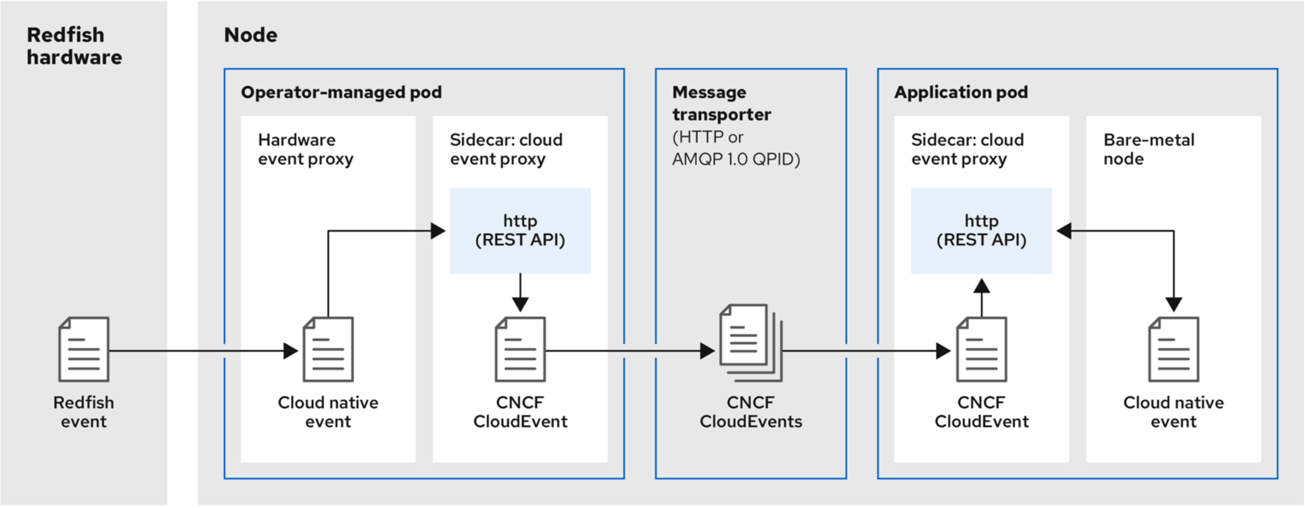
The Bare Metal Event Relay enables applications running on bare-metal clusters to respond quickly to Redfish hardware changes and failures such as breaches of temperature thresholds, fan failure, disk loss, power outages, and memory failure. These hardware events are delivered using an HTTP transport or AMQP mechanism. The latency of the messaging service is between 10 to 20 milliseconds.

The Bare Metal Event Relay provides a publish-subscribe service for the hardware events. Applications can use a REST API to subscribe to the events. The Bare Metal Event Relay supports hardware that complies with Redfish OpenAPI v1.8 or later.

9.2.1. Bare Metal Event Relay data flow

The following figure illustrates an example bare-metal events data flow:

Figure 9.1. Bare Metal Event Relay data flow



319_OpenShift_0323

9.2.1.1. Operator-managed pod

The Operator uses custom resources to manage the pod containing the Bare Metal Event Relay and its components using the **HardwareEvent** CR.

9.2.1.2. Bare Metal Event Relay

At startup, the Bare Metal Event Relay queries the Redfish API and downloads all the message registries, including custom registries. The Bare Metal Event Relay then begins to receive subscribed events from the Redfish hardware.

The Bare Metal Event Relay enables applications running on bare-metal clusters to respond quickly to Redfish hardware changes and failures such as breaches of temperature thresholds, fan failure, disk loss, power outages, and memory failure. The events are reported using the **HardwareEvent** CR.

9.2.1.3. Cloud native event

Cloud native events (CNE) is a REST API specification for defining the format of event data.

9.2.1.4. CNCF CloudEvents

CloudEvents is a vendor-neutral specification developed by the Cloud Native Computing Foundation (CNCF) for defining the format of event data.

9.2.1.5. HTTP transport or AMQP dispatch router

The HTTP transport or AMQP dispatch router is responsible for the message delivery service between publisher and subscriber.

NOTE

HTTP transport is the default transport for PTP and bare-metal events. Use HTTP transport instead of AMQP for PTP and bare-metal events where possible. AMQ Interconnect is EOL from 30 June 2024. Extended life cycle support (ELS) for AMQ Interconnect ends 29 November 2029. For more information see, Red Hat AMQ Interconnect support status.

9.2.1.6. Cloud event proxy sidecar

The cloud event proxy sidecar container image is based on the O-RAN API specification and provides a publish-subscribe event framework for hardware events.

9.2.2. Redfish message parsing service

In addition to handling Redfish events, the Bare Metal Event Relay provides message parsing for events without a **Message** property. The proxy downloads all the Redfish message registries including vendor specific registries from the hardware when it starts. If an event does not contain a **Message** property, the proxy uses the Redfish message registries to construct the **Message** and **Resolution** properties and add them to the event before passing the event to the cloud events framework. This service allows Redfish events to have smaller message size and lower transmission latency.

9.2.3. Installing the Bare Metal Event Relay using the CLI

As a cluster administrator, you can install the Bare Metal Event Relay Operator by using the CLI.

Prerequisites

- A cluster that is installed on bare-metal hardware with nodes that have a RedFish-enabled Baseboard Management Controller (BMC).
- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

Procedure

1. Create a namespace for the Bare Metal Event Relay.
   a. Save the following YAML in the **bare-metal-events-namespace.yaml** file:

```
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-bare-metal-events
  labels:
    name: openshift-bare-metal-events
    openshift.io/cluster-monitoring: "true"
```

   b. Create the **Namespace** CR:

```
$ oc create -f bare-metal-events-namespace.yaml
```

2. Create an Operator group for the Bare Metal Event Relay Operator.

a. Save the following YAML in the **bare-metal-events-operatorgroup.yaml** file:

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: bare-metal-event-relay-group
  namespace: openshift-bare-metal-events
spec:
  targetNamespaces:
  - openshift-bare-metal-events
```

b. Create the **OperatorGroup** CR:

```
$ oc create -f bare-metal-events-operatorgroup.yaml
```

3. Subscribe to the Bare Metal Event Relay.

a. Save the following YAML in the **bare-metal-events-sub.yaml** file:

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: bare-metal-event-relay-subscription
  namespace: openshift-bare-metal-events
spec:
  channel: "stable"
  name: bare-metal-event-relay
  source: redhat-operators
  sourceNamespace: openshift-marketplace
```

b. Create the **Subscription** CR:

```
$ oc create -f bare-metal-events-sub.yaml
```

Verification

To verify that the Bare Metal Event Relay Operator is installed, run the following command:

```
$ oc get csv -n openshift-bare-metal-events -o custom-
columns=Name:.metadata.name,Phase:.status.phase
```

9.2.4. Installing the Bare Metal Event Relay using the web console

As a cluster administrator, you can install the Bare Metal Event Relay Operator using the web console.

Prerequisites

- A cluster that is installed on bare-metal hardware with nodes that have a RedFish-enabled Baseboard Management Controller (BMC).

- Log in as a user with **cluster-admin** privileges.

Procedure

1. Install the Bare Metal Event Relay using the OpenShift Container Platform web console:

    a. In the OpenShift Container Platform web console, click **Operators** → **OperatorHub**.

    b. Choose **Bare Metal Event Relay**from the list of available Operators, and then click **Install**.

    c. On the **Install Operator** page, select or create a **Namespace**, select **openshift-bare-metal-events**, and then click **Install**.

Verification

Optional: You can verify that the Operator installed successfully by performing the following check:

1. Switch to the **Operators** → **Installed Operators** page.

2. Ensure that **Bare Metal Event Relay**is listed in the 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 project namespace.

9.3. INSTALLING THE AMQ MESSAGING BUS

To pass Redfish bare-metal event notifications between publisher and subscriber on a node, you can install and configure an AMQ messaging bus to run locally on the node. You do this by installing the AMQ Interconnect Operator for use in the cluster.

> **NOTE**
>
> HTTP transport is the default transport for PTP and bare-metal events. Use HTTP transport instead of AMQP for PTP and bare-metal events where possible. AMQ Interconnect is EOL from 30 June 2024. Extended life cycle support (ELS) for AMQ Interconnect ends 29 November 2029. For more information see, Red Hat AMQ Interconnect support status.

Prerequisites

- Install the OpenShift Container Platform CLI (**oc**).

- Log in as a user with **cluster-admin** privileges.

Procedure

- Install the AMQ Interconnect Operator to its own **amq-interconnect** namespace. See Installing the AMQ Interconnect Operator.

Verification

1. Verify that the AMQ Interconnect Operator is available and the required pods are running:

   ```
   $ oc get pods -n amq-interconnect
   ```

   Example output

   ```
   NAME                                  READY   STATUS    RESTARTS   AGE
   amq-interconnect-645db76c76-k8ghs      1/1     Running   0          23h
   interconnect-operator-5cb5fc7cc-4v7qm  1/1     Running   0          23h
   ```

2. Verify that the required **bare-metal-event-relay** bare-metal event producer pod is running in the **openshift-bare-metal-events** namespace:

   ```
   $ oc get pods -n openshift-bare-metal-events
   ```

   Example output

   ```
   NAME                                                    READY   STATUS    RESTARTS   AGE
   hw-event-proxy-operator-controller-manager-74d5649b7c-dzgtl    2/2     Running   0
   25s
   ```

9.4. SUBSCRIBING TO REDFISH BMC BARE-METAL EVENTS FOR A CLUSTER NODE

You can subscribe to Redfish BMC events generated on a node in your cluster by creating a **BMCEventSubscription** custom resource (CR) for the node, creating a **HardwareEvent** CR for the event, and creating a **Secret** CR for the BMC.

9.4.1. Subscribing to bare-metal events

You can configure the baseboard management controller (BMC) to send bare-metal events to subscribed applications running in an OpenShift Container Platform cluster. Example Redfish bare-metal events include an increase in device temperature, or removal of a device. You subscribe applications to bare-metal events using a REST API.

IMPORTANT

You can only create a **BMCEventSubscription** custom resource (CR) for physical hardware that supports Redfish and has a vendor interface set to **redfish** or **idrac-redfish**.

NOTE

Use the **BMCEventSubscription** CR to subscribe to predefined Redfish events. The Redfish standard does not provide an option to create specific alerts and thresholds. For example, to receive an alert event when an enclosure’s temperature exceeds 40° Celsius, you must manually configure the event according to the vendor’s recommendations.

Perform the following procedure to subscribe to bare-metal events for the node using a **BMCEventSubscription** CR.

Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.
- Get the user name and password for the BMC.
- Deploy a bare-metal node with a Redfish-enabled Baseboard Management Controller (BMC) in your cluster, and enable Redfish events on the BMC.

> **NOTE**
>
> Enabling Redfish events on specific hardware is outside the scope of this information. For more information about enabling Redfish events for your specific hardware, consult the BMC manufacturer documentation.

Procedure

1. Confirm that the node hardware has the Redfish **EventService** enabled by running the following **curl** command:

```
$ curl https://<bmc_ip_address>/redfish/v1/EventService --insecure -H 'Content-Type:
application/json' -u "<bmc_username>:<password>"
```

where:

bmc_ip_address

is the IP address of the BMC where the Redfish events are generated.

Example output

```
{
  "@odata.context": "/redfish/v1/$metadata#EventService.EventService",
  "@odata.id": "/redfish/v1/EventService",
  "@odata.type": "#EventService.v1_0_2.EventService",
  "Actions": {
    "#EventService.SubmitTestEvent": {
      "EventType@Redfish.AllowableValues": ["StatusChange", "ResourceUpdated",
"ResourceAdded", "ResourceRemoved", "Alert"],
      "target": "/redfish/v1/EventService/Actions/EventService.SubmitTestEvent"
    }
  },
  "DeliveryRetryAttempts": 3,
  "DeliveryRetryIntervalSeconds": 30,
  "Description": "Event Service represents the properties for the service",
  "EventTypesForSubscription": ["StatusChange", "ResourceUpdated", "ResourceAdded",
"ResourceRemoved", "Alert"],
  "EventTypesForSubscription@odata.count": 5,
  "Id": "EventService",
  "Name": "Event Service",
  "ServiceEnabled": true,
  "Status": {
    "Health": "OK",
    "HealthRollup": "OK",
      "State": "Enabled"
    },
    "Subscriptions": {
      "@odata.id": "/redfish/v1/EventService/Subscriptions"
    }
  }
```

2. Get the Bare Metal Event Relay service route for the cluster by running the following command:

```
$ oc get route -n openshift-bare-metal-events
```

Example output

```
NAME          HOST/PORT   PATH                                                    SERVICES
PORT   TERMINATION   WILDCARD
hw-event-proxy            hw-event-proxy-openshift-bare-metal-events.apps.compute-
1.example.com   hw-event-proxy-service   9087   edge         None
```

3. Create a **BMCEventSubscription** resource to subscribe to the Redfish events:

   a. Save the following YAML in the **bmc_sub.yaml** file:

   ```
   apiVersion: metal3.io/v1alpha1
   kind: BMCEventSubscription
   metadata:
     name: sub-01
     namespace: openshift-machine-api
   spec:
      hostName: <hostname> 1
      destination: <proxy_service_url> 2
      context: ''
   ```

   1 Specifies the name or UUID of the worker node where the Redfish events are generated.

   2 Specifies the bare-metal event proxy service, for example, **https://hw-event-proxy-openshift-bare-metal-events.apps.compute-1.example.com/webhook**.

   b. Create the **BMCEventSubscription** CR:

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

4. Optional: To delete the BMC event subscription, run the following command:

```
$ oc delete -f bmc_sub.yaml
```

5. Optional: To manually create a Redfish event subscription without creating a **BMCEventSubscription** CR, run the following **curl** command, specifying the BMC username and password.

```
$ curl -i -k -X POST -H "Content-Type: application/json"  -d '{"Destination":
"https://<proxy_service_url>", "Protocol" : "Redfish", "EventTypes": ["Alert"], "Context":
"root"}' -u <bmc_username>:<password>
```

```
'https://<bmc_ip_address>/redfish/v1/EventService/Subscriptions' –v
```

where:

proxy_service_url

is the bare-metal event proxy service, for example, **https://hw-event-proxy-openshift-bare-metal-events.apps.compute-1.example.com/webhook**.

bmc_ip_address

is the IP address of the BMC where the Redfish events are generated.

Example output

```
HTTP/1.1 201 Created
Server: AMI MegaRAC Redfish Service
Location: /redfish/v1/EventService/Subscriptions/1
Allow: GET, POST
Access-Control-Allow-Origin: *
Access-Control-Expose-Headers: X-Auth-Token
Access-Control-Allow-Headers: X-Auth-Token
Access-Control-Allow-Credentials: true
Cache-Control: no-cache, must-revalidate
Link: <http://redfish.dmtf.org/schemas/v1/EventDestination.v1_6_0.json>; rel=describedby
Link: <http://redfish.dmtf.org/schemas/v1/EventDestination.v1_6_0.json>
Link: </redfish/v1/EventService/Subscriptions>; path=
ETag: "1651135676"
Content-Type: application/json; charset=UTF-8
OData-Version: 4.0
Content-Length: 614
Date: Thu, 28 Apr 2022 08:47:57 GMT
```

9.4.2. Querying Redfish bare-metal event subscriptions with curl

Some hardware vendors limit the amount of Redfish hardware event subscriptions. You can query the number of Redfish event subscriptions by using **curl**.

Prerequisites

- Get the user name and password for the BMC.
- Deploy a bare-metal node with a Redfish-enabled Baseboard Management Controller (BMC) in your cluster, and enable Redfish hardware events on the BMC.

Procedure

1. Check the current subscriptions for the BMC by running the following **curl** command:

```
$ curl --globoff -H "Content-Type: application/json" -k -X GET --user <bmc_username>:
<password> https://<bmc_ip_address>/redfish/v1/EventService/Subscriptions
```

where:

bmc_ip_address

is the IP address of the BMC where the Redfish events are generated.

Example output

```
% Total % Received % Xferd Average Speed Time Time Time Current
Dload Upload Total Spent Left Speed
100 435 100 435 0 0 399 0 0:00:01 0:00:01 --:--:-- 399
{
  "@odata.context":
"/redfish/v1/$metadata#EventDestinationCollection.EventDestinationCollection",
  "@odata.etag": ""
  1651137375 "",
  "@odata.id": "/redfish/v1/EventService/Subscriptions",
  "@odata.type": "#EventDestinationCollection.EventDestinationCollection",
  "Description": "Collection for Event Subscriptions",
  "Members": [
  {
    "@odata.id": "/redfish/v1/EventService/Subscriptions/1"
  }],
  "Members@odata.count": 1,
  "Name": "Event Subscriptions Collection"
}
```

In this example, a single subscription is configured: **/redfish/v1/EventService/Subscriptions/1**.

2. Optional: To remove the **/redfish/v1/EventService/Subscriptions/1** subscription with **curl**, run the following command, specifying the BMC username and password:

```
$ curl --globoff -L -w "%{http_code} %{url_effective}\n" -k -u <bmc_username>:<password >-
H "Content-Type: application/json" -d '{}' -X DELETE
https://<bmc_ip_address>/redfish/v1/EventService/Subscriptions/1
```

where:

bmc_ip_address

is the IP address of the BMC where the Redfish events are generated.

9.4.3. Creating the bare-metal event and Secret CRs

To start using bare-metal events, create the **HardwareEvent** custom resource (CR) for the host where the Redfish hardware is present. Hardware events and faults are reported in the **hw-event-proxy** logs.

Prerequisites

- You have installed the OpenShift Container Platform CLI (**oc**).
- You have logged in as a user with **cluster-admin** privileges.
- You have installed the Bare Metal Event Relay.
- You have created a **BMCEventSubscription** CR for the BMC Redfish hardware.

Procedure

1. Create the **HardwareEvent** custom resource (CR):

NOTE

Multiple **HardwareEvent** resources are not permitted.

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

```
apiVersion: "event.redhat-cne.org/v1alpha1"
kind: "HardwareEvent"
metadata:
  name: "hardware-event"
spec:
  nodeSelector:
    node-role.kubernetes.io/hw-event: "" 1
  logLevel: "debug" 2
  msgParserTimeout: "10" 3
```

1 Required. Use the **nodeSelector** field to target nodes with the specified label, for example, **node-role.kubernetes.io/hw-event: ""**.

NOTE

In OpenShift Container Platform 4.12 or later, you do not need to set the **spec.transportHost** field in the **HardwareEvent** resource when you use HTTP transport for bare-metal events. Set **transportHost** only when you use AMQP transport for bare-metal events.

2 Optional. The default value is **debug**. Sets the log level in **hw-event-proxy** logs. The following log levels are available: **fatal**, **error**, **warning**, **info**, **debug**, **trace**.

3 Optional. Sets the timeout value in milliseconds for the Message Parser. If a message parsing request is not responded to within the timeout duration, the original hardware event message is passed to the cloud native event framework. The default value is 10.

b. Apply the **HardwareEvent** CR in the cluster:

```
$ oc create -f hardware-event.yaml
```

2. Create a BMC username and password **Secret** CR that enables the hardware events proxy to access the Redfish message registry for the bare-metal host.

a. Save the following YAML in the **hw-event-bmc-secret.yaml** file:

```
apiVersion: v1
kind: Secret
metadata:
  name: redfish-basic-auth
type: Opaque
stringData: 1
  username: <bmc_username>
  password: <bmc_password>
  # BMC host DNS or IP address
  hostaddr: <bmc_host_ip_address>
```

1 Enter plain text values for the various items under **stringData**.

b. Create the **Secret** CR:

```
$ oc create -f hw-event-bmc-secret.yaml
```

Additional resources

- Persistent storage using local volumes

9.5. SUBSCRIBING APPLICATIONS TO BARE-METAL EVENTS REST API REFERENCE

Use the bare-metal events REST API to subscribe an application to the bare-metal events that are generated on the parent node.

Subscribe applications to Redfish events by using the resource address **/cluster/node/<node_name>/redfish/event**, where **<node_name>** is the cluster node running the application.

Deploy your **cloud-event-consumer** application container and **cloud-event-proxy** sidecar container in a separate application pod. The **cloud-event-consumer** application subscribes to the **cloud-event-proxy** container in the application pod.

Use the following API endpoints to subscribe the **cloud-event-consumer** application to Redfish events posted by the **cloud-event-proxy** container at **http://localhost:8089/api/ocloudNotifications/v1/** in the application pod:

- **/api/ocloudNotifications/v1/subscriptions**
  - **POST**: Creates a new subscription
  - **GET**: Retrieves a list of subscriptions
- **/api/ocloudNotifications/v1/subscriptions/<subscription_id>**
  - **PUT**: Creates a new status ping request for the specified subscription ID
- **/api/ocloudNotifications/v1/health**
  - **GET**: Returns the health status of **ocloudNotifications** API

NOTE

9089 is the default port for the **cloud-event-consumer** container deployed in the application pod. You can configure a different port for your application as required.

api/ocloudNotifications/v1/subscriptions

HTTP method

GET api/ocloudNotifications/v1/subscriptions

Description

Returns a list of subscriptions. If subscriptions exist, a **200 OK** status code is returned along with the list of subscriptions.

Example API response

```
[
 {
  "id": "ca11ab76-86f9-428c-8d3a-666c24e34d32",
  "endpointUri": "http://localhost:9089/api/ocloudNotifications/v1/dummy",
  "uriLocation": "http://localhost:8089/api/ocloudNotifications/v1/subscriptions/ca11ab76-86f9-428c-8d3a-666c24e34d32",
  "resource": "/cluster/node/openshift-worker-0.openshift.example.com/redfish/event"
 }
]
```

HTTP method
POST api/ocloudNotifications/v1/subscriptions

Description
Creates a new subscription. If a subscription is successfully created, or if it already exists, a **201 Created** status code is returned.

Table 9.1. Query parameters

Parameter	Type
subscription	data

Example payload

```
{
  "uriLocation": "http://localhost:8089/api/ocloudNotifications/v1/subscriptions",
  "resource": "/cluster/node/openshift-worker-0.openshift.example.com/redfish/event"
}
```

api/ocloudNotifications/v1/subscriptions/<subscription_id>

HTTP method
GET api/ocloudNotifications/v1/subscriptions/<subscription_id>

Description
Returns details for the subscription with ID **<subscription_id>**

Table 9.2. Query parameters

Parameter	Type
<subscription_id>	string

Example API response

```
{
  "id":"ca11ab76-86f9-428c-8d3a-666c24e34d32",
  "endpointUri":"http://localhost:9089/api/ocloudNotifications/v1/dummy",
  "uriLocation":"http://localhost:8089/api/ocloudNotifications/v1/subscriptions/ca11ab76-86f9-428c-
8d3a-666c24e34d32",
  "resource":"/cluster/node/openshift-worker-0.openshift.example.com/redfish/event"
}
```

api/ocloudNotifications/v1/health/

HTTP method

GET api/ocloudNotifications/v1/health/

Description

Returns the health status for the **ocloudNotifications** REST API.

Example API response

```
OK
```

9.6. MIGRATING CONSUMER APPLICATIONS TO USE HTTP TRANSPORT FOR PTP OR BARE-METAL EVENTS

If you have previously deployed PTP or bare-metal events consumer applications, you need to update the applications to use HTTP message transport.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have logged in as a user with **cluster-admin** privileges.
- You have updated the PTP Operator or Bare Metal Event Relay to version 4.12 or later which uses HTTP transport by default.

Procedure

1. Update your events consumer application to use HTTP transport. Set the **http-event-publishers** variable for the cloud event sidecar deployment.
   For example, in a cluster with PTP events configured, the following YAML snippet illustrates a cloud event sidecar deployment:

```
containers:
  - name: cloud-event-sidecar
    image: cloud-event-sidecar
    args:
      - "--metrics-addr=127.0.0.1:9091"
      - "--store-path=/store"
      - "--transport-host=consumer-events-subscription-service.cloud-
events.svc.cluster.local:9043"
      - "--http-event-publishers=ptp-event-publisher-service-NODE_NAME.openshift-
ptp.svc.cluster.local:9043" 1
      - "--api-port=8089"
```

1 The PTP Operator automatically resolves **NODE_NAME** to the host that is generating the PTP events. For example, **compute-1.example.com**.

In a cluster with bare-metal events configured, set the **http-event-publishers** field to **hw-event-publisher-service.openshift-bare-metal-events.svc.cluster.local:9043** in the cloud event sidecar deployment CR.

2. Deploy the **consumer-events-subscription-service** service alongside the events consumer application. For example:

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    prometheus.io/scrape: "true"
    service.alpha.openshift.io/serving-cert-secret-name: sidecar-consumer-secret
  name: consumer-events-subscription-service
  namespace: cloud-events
  labels:
    app: consumer-service
spec:
  ports:
    - name: sub-port
      port: 9043
  selector:
    app: consumer
  clusterIP: None
  sessionAffinity: None
  type: ClusterIP
```

CHAPTER 10. WHAT HUGE PAGES DO AND HOW THEY ARE CONSUMED BY APPLICATIONS

10.1. 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.

10.2. HOW HUGE PAGES ARE CONSUMED BY APPS

Nodes must pre-allocate huge pages in order for the node to report its huge page capacity. A node can only pre-allocate huge pages for a single size.

Huge pages can be consumed through container-level resource requirements using the resource name **hugepages-<size>**, where size is the most compact binary notation using integer values supported on a particular node. For example, if a node supports 2048KiB page sizes, it exposes a schedulable resource **hugepages-2Mi**. Unlike CPU or memory, huge pages do not support over-commitment.

```
apiVersion: v1
kind: Pod
metadata:
  generateName: hugepages-volume-
spec:
  containers:
  - securityContext:
      privileged: true
    image: rhel7:latest
    command:
    - sleep
    - inf
    name: example
    volumeMounts:
    - mountPath: /dev/hugepages
      name: hugepage
    resources:
      limits:
```

```
   hugepages-2Mi: 100Mi 1
   memory: "1Gi"
   cpu: "1"
volumes:
- name: hugepage
 emptyDir:
  medium: HugePages
```

1 Specify the amount of memory for **hugepages** as the exact amount to be allocated. Do not specify this value as the amount of memory for **hugepages** multiplied by the size of the page. For example, given a huge page size of 2MB, if you want to use 100MB of huge-page-backed RAM for your application, then you would allocate 50 huge pages. OpenShift Container Platform handles the math for you. As in the above example, you can specify **100MB** directly.

Allocating huge pages of a specific size

Some platforms support multiple huge page sizes. To allocate huge pages of a specific size, precede the huge pages boot command parameters with a huge page size selection parameter **hugepagesz=<size>**. The **<size>** value must be specified in bytes with an optional scale suffix [**kKmMgG**]. The default huge page size can be defined with the **default_hugepagesz=<size>** boot parameter.

Huge page requirements

- Huge page requests must equal the limits. This is the default if limits are specified, but requests are not.
- Huge pages are isolated at a pod scope. Container isolation is planned in a future iteration.
- **EmptyDir** volumes backed by huge pages must not consume more huge page memory than the pod request.
- Applications that consume huge pages via **shmget()** with **SHM_HUGETLB** must run with a supplemental group that matches ***proc/sys/vm/hugetlb_shm_group***.

10.3. CONSUMING HUGE PAGES RESOURCES USING THE DOWNWARD API

You can use the Downward API to inject information about the huge pages resources that are consumed by a container.

You can inject the resource allocation as environment variables, a volume plugin, or both. Applications that you develop and run in the container can determine the resources that are available by reading the environment variables or files in the specified volumes.

Procedure

1. Create a **hugepages-volume-pod.yaml** file that is similar to the following example:

```
apiVersion: v1
kind: Pod
metadata:
 generateName: hugepages-volume-
 labels:
  app: hugepages-example
```

```
spec:
  containers:
  - securityContext:
      capabilities:
        add: [ "IPC_LOCK" ]
    image: rhel7:latest
    command:
    - sleep
    - inf
    name: example
    volumeMounts:
    - mountPath: /dev/hugepages
      name: hugepage
    - mountPath: /etc/podinfo
      name: podinfo
    resources:
      limits:
        hugepages-1Gi: 2Gi
        memory: "1Gi"
        cpu: "1"
      requests:
        hugepages-1Gi: 2Gi
    env:
    - name: REQUESTS_HUGEPAGES_1GI <.>
      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_HUGEPAGES_1GI
```

Example output

```
REQUESTS_HUGEPAGES_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

10.4. CONFIGURING HUGE PAGES AT BOOT TIME

Nodes must pre-allocate huge pages used in an OpenShift Container Platform cluster. There are two ways of reserving huge pages: at boot time and at run time. Reserving at boot time increases the possibility of success because the memory has not yet been significantly fragmented. The Node Tuning Operator currently supports boot time allocation of huge pages on specific nodes.

Procedure

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

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

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

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

```
apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
  name: hugepages 1
  namespace: openshift-cluster-node-tuning-operator
spec:
  profile: 2
  - data: |
      [main]
      summary=Boot time configuration for hugepages
      include=openshift-node
      [bootloader]
      cmdline_openshift_node_hugepages=hugepagesz=2M hugepages=50 3
  name: openshift-node-hugepages

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

1 Set the **name** of the Tuned resource to **hugepages**.

2 Set the **profile** section to allocate huge pages.

3 Note the order of parameters is important as some platforms support huge pages of various sizes.

4 Enable machine config pool based matching.

3. Create the Tuned **hugepages** object

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

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

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

5. Create the machine config pool:

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

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

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

NOTE

The TuneD bootloader plugin only supports Red Hat Enterprise Linux CoreOS (RHCOS) worker nodes.

10.5. 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 11. LOW LATENCY TUNING

11.1. 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.

NOTE

Currently, disabling CPU load balancing is not supported by cgroup v2. As a result, you might not get the desired behavior from performance profiles if you have cgroup v2 enabled. Enabling cgroup v2 is not recommended if you are using performance profiles.

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.

11.1.1. 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

11.2. 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.

11.2.1. 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.

11.2.2. Provisioning a worker with real-time capabilities

1. Optional: Add a node to the OpenShift Container Platform cluster. See Setting BIOS parameters for system tuning.

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 that 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.

11.2.3. 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://1.25.0-99.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   v1.25.0
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://1.25.0-99.rhaos4.10.gitc3131de.el8
[...]
```

11.2.4. 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.

11.2.5. 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.

11.2.6. 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
```

NOTE

Currently, disabling CPU load balancing is not supported with cgroup v2.

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.

11.2.7. 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 .

11.2.8. 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.

11.2.9. 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" (1)
  nodeSelector:
    node-role.kubernetes.io/worker-cnf: ""
  numa:
    topologyPolicy: single-numa-node
  realTimeKernel:
    enabled: true
```

   (1) 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
```

–

11.2.10. Optional: Power saving configurations

You can enable power savings for a node that has low priority workloads that are colocated with high priority workloads without impacting the latency or throughput of the high priority workloads. Power saving is possible without modifications to the workloads themselves.

> **IMPORTANT**
>
> The feature is supported on Intel Ice Lake and later generations of Intel CPUs. The capabilities of the processor might impact the latency and throughput of the high priority workloads.

When you configure a node with a power saving configuration, you must configure high priority workloads with performance configuration at the pod level, which means that the configuration applies to all the cores used by the pod.

By disabling P-states and C-states at the pod level, you can configure high priority workloads for best performance and lowest latency.

Table 11.1. Configuration for high priority workloads

Annotation	Description

`annotations:`
`  cpu-c-states.crio.io: "disable"`
`  cpu-freq-governor.crio.io: "<governor>"`
	Provides the best performance for a pod by disabling C-states and specifying the governor type for CPU scaling. The **performance** governor is recommended for high priority workloads.

Prerequisites

- You enabled C-states and OS-controlled P-states in the BIOS

Procedure

1. Generate a **PerformanceProfile** with **per-pod-power-management** set to **true**:

```
$ podman run --entrypoint performance-profile-creator -v \
/must-gather:/must-gather:z registry.redhat.io/openshift4/ose-cluster-node-tuning-operator:v4.12 \
--mcp-name=worker-cnf --reserved-cpu-count=20 --rt-kernel=true \
--split-reserved-cpus-across-numa=false --topology-manager-policy=single-numa-node \
--must-gather-dir-path /must-gather -power-consumption-mode=low-latency \ 1
--per-pod-power-management=true > my-performance-profile.yaml
```

1 The **power-consumption-mode** must be **default** or **low-latency** when the **per-pod-power-management** is set to **true**.

Example PerformanceProfile with perPodPowerManagement

–

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: performance
spec:
  [.....]
  workloadHints:
    realTime: true
    highPowerConsumption: false
    perPodPowerManagement: true
```

2. Set the default **cpufreq** governor as an additional kernel argument in the **PerformanceProfile** custom resource (CR):

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: performance
spec:
  ...
  additionalKernelArgs:
  - cpufreq.default_governor=schedutil 1
```

1 Using the **schedutil** governor is recommended, however, you can use other governors such as the **ondemand** or **powersave** governors.

3. Set the maximum CPU frequency in the **TunedPerformancePatch** CR:

```
spec:
 profile:
 - data: |
   [sysfs]
   /sys/devices/system/cpu/intel_pstate/max_perf_pct = <x> 1
```

1 The **max_perf_pct** controls the maximum frequency the **cpufreq** driver is allowed to set as a percentage of the maximum supported cpu frequency. This value applies to all CPUs. You can check the maximum supported frequency in **/sys/devices/system/cpu/cpu0/cpufreq/cpuinfo_max_freq**. As a starting point, you can use a percentage that caps all CPUs at the **All Cores Turbo** frequency. The **All Cores Turbo** frequency is the frequency that all cores will run at when the cores are all fully occupied.

4. Add the desired annotations to your high priority workload pods. The annotations override the **default** settings.

Example high priority workload annotation

```
apiVersion: v1
kind: Pod
metadata:
 ...
 annotations:
  ...
```

```
    cpu-c-states.crio.io: "disable"
    cpu-freq-governor.crio.io: "<governor>"
    ...
  ...
spec:
  ...
  runtimeClassName: performance-<profile_name>
  ...
```

5. Restart the pods.

Additional resources

- For more information about recommended firmware configuration, see Recommended firmware configuration for vDU cluster hosts.

11.2.11. 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.

11.2.11.1. 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: v1
kind: Pod
metadata:
  annotations:
      cpu-quota.crio.io: "disable"
spec:
    runtimeClassName: performance-<profile_name>
...
```

NOTE

Only disable CPU CFS quota when the CPU manager static policy is enabled and for pods with guaranteed QoS that use whole CPUs. Otherwise, disabling CPU CFS quota can affect the performance of other containers in the cluster.

11.2.11.2. 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
...
```

11.2.11.3. 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>
...
```

11.2.12. 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.

11.2.12.1. 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**.

11.2.12.1.1. 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.

11.2.12.1.2. 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**.

11.3. 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: "4-15" ①
  reserved: "0-3" 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. Set an even number of isolated CPUs to enable the pods to run without errors when hyperthreading is enabled.

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 .

11.3.1. 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-###:  ###
```

11.3.2. 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
```

11.3.3. Configuring a node for IRQ dynamic load balancing

Configure a cluster node for IRQ dynamic load balancing to control which cores can receive device interrupt requests (IRQ).

Prerequisites

- For core isolation, all server hardware components must support IRQ affinity. To check if the hardware components of your server support IRQ affinity, view the server's hardware specifications or contact your hardware provider.

Procedure

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:v4.12"
    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. Log in to 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
```

11.3.4. About support of IRQ affinity setting

Some IRQ controllers lack support for IRQ affinity setting and will always expose all online CPUs as the IRQ mask. These IRQ controllers effectively run on CPU 0.

The following are examples of drivers and hardware that Red Hat are aware lack support for IRQ affinity setting. The list is, by no means, exhaustive:

- Some RAID controller drivers, such as **megaraid_sas**
- Many non-volatile memory express (NVMe) drivers

- Some LAN on motherboard (LOM) network controllers
- The driver uses **managed_irqs**

> **NOTE**
>
> The reason they do not support IRQ affinity setting might be associated with factors such as the type of processor, the IRQ controller, or the circuitry connections in the motherboard.

If the effective affinity of any IRQ is set to an isolated CPU, it might be a sign of some hardware or driver not supporting IRQ affinity setting. To find the effective affinity, log in to the host and run the following command:

```
$ find /proc/irq -name effective_affinity -printf "%p: " -exec cat {} \;
```

Example output

```
/proc/irq/0/effective_affinity: 1
/proc/irq/1/effective_affinity: 8
/proc/irq/2/effective_affinity: 0
/proc/irq/3/effective_affinity: 1
/proc/irq/4/effective_affinity: 2
/proc/irq/5/effective_affinity: 1
/proc/irq/6/effective_affinity: 1
/proc/irq/7/effective_affinity: 1
/proc/irq/8/effective_affinity: 1
/proc/irq/9/effective_affinity: 2
/proc/irq/10/effective_affinity: 1
/proc/irq/11/effective_affinity: 1
/proc/irq/12/effective_affinity: 4
/proc/irq/13/effective_affinity: 1
/proc/irq/14/effective_affinity: 1
/proc/irq/15/effective_affinity: 1
/proc/irq/24/effective_affinity: 2
/proc/irq/25/effective_affinity: 4
/proc/irq/26/effective_affinity: 2
/proc/irq/27/effective_affinity: 1
/proc/irq/28/effective_affinity: 8
/proc/irq/29/effective_affinity: 4
/proc/irq/30/effective_affinity: 4
/proc/irq/31/effective_affinity: 8
/proc/irq/32/effective_affinity: 8
/proc/irq/33/effective_affinity: 1
/proc/irq/34/effective_affinity: 2
```

Some drivers use **managed_irqs**, whose affinity is managed internally by the kernel and userspace cannot change the affinity. In some cases, these IRQs might be assigned to isolated CPUs. For more information about **managed_irqs**, see Affinity of managed interrupts cannot be changed even if they target isolated CPU.

11.3.5. 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.

11.3.5.1. 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.

11.3.6. 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. If the workload hint is configured manually and the **realTime** workload hint is not explicitly set then it defaults to **true**.

Performance Profile creator setting	Hint	Environment	Description

Default	`workloadHints:`
	`highPowerConsumption: false`
	`realTime: false`
		High throughput cluster without latency requirements	Performance achieved through CPU partitioning only.

Low-latency	`workloadHints:`
	`highPowerConsumption: false`
	`realTime: true`
		Regional datacenters	Both energy savings and low-latency are desirable: compromise between power management, latency and throughput.

Ultra-low-latency	`workloadHints:`
	`highPowerConsumption: true`
	`realTime: true`
		Far edge clusters, latency critical workloads	Optimized for absolute minimal latency and maximum determinism at the cost of increased power consumption.

Performance Profile creator setting	Hint	Environment	Description
Per-pod power management	workloadHints: realTime: true highPowerConsumption: false perPodPowerManagement: true	Critical and non-critical workloads	Allows for power management per pod.

Additional resources

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

11.3.7. Configuring workload hints manually

Procedure

1. 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.

2. 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.

NOTE

When the **realTime** workload hint flag is set to **true** in a performance profile, add the **cpu-quota.crio.io: disable** annotation to every guaranteed pod with pinned CPUs. This annotation is necessary to prevent the degradation of the process performance within the pod. If the **realTime** workload hint is not explicitly set then it defaults to **true**.

Additional resources

- For information about reducing CPU throttling for individual guaranteed pods, see Disabling CPU CFS quota.

11.3.8. 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 11.2. 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 4.12 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

11.4. 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 using a PerformanceProfile, the amount of queues can be reduced to the number of reserved CPUs.

11.4.1. 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: 3-51,55-103
    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: 3-51,55-103
    reserved: 0-2,52-54
  net:
    userLevelNetworking: true
    devices:
    - interfaceName: "eth0"
    - interfaceName: "eth1"
    - vendorID: "0x1af4"
      deviceID: "0x1000"
  nodeSelector:
    node-role.kubernetes.io/worker-cnf: ""
```

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: 3-51,55-103
    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: 3-51,55-103
    reserved: 0-2,52-54
  net:
    userLevelNetworking: true
    devices:
    - interfaceName: "!eno1"
  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: 3-51,55-103
    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 .

11.4.2. 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

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.

11.4.3. 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
```

11.5. 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.

11.5.1. 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
```

11.6. 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.

11.6.1. 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.

11.6.2. 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:v4.12.
```

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 and later, this functionality is part of the Node Tuning Operator. However, you must still use the **performance-addon-operator-must-gather** image when running the **must-gather** command.

11.6.3. 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 \ ❶

 --image=registry.redhat.io/openshift4/performance-addon-operator-must-gather-rhel8:v4.12
❷
```

❶ The default OpenShift Container Platform **must-gather** image.

❷ 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/ ❶
```

❶ 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 12. PERFORMING LATENCY TESTS FOR PLATFORM VERIFICATION

You can use the Cloud-native Network Functions (CNF) tests image to run latency tests on a CNF-enabled OpenShift Container Platform cluster, where all the components required for running CNF workloads are installed. Run the latency tests to validate node tuning for your workload.

The **cnf-tests** container image is available at **registry.redhat.io/openshift4/cnf-tests-rhel8:v4.12**.

IMPORTANT

The **cnf-tests** image also includes several tests that are not supported by Red Hat at this time. Only the latency tests are supported by Red Hat.

12.1. PREREQUISITES FOR RUNNING LATENCY TESTS

Your cluster must meet the following requirements before you can run the latency tests:

1. You have configured a performance profile with the Node Tuning Operator.
2. You have applied all the required CNF configurations in the cluster.
3. You have a pre-existing **MachineConfigPool** CR applied in the cluster. The default worker pool is **worker-cnf**.

Additional resources

- For more information about creating the cluster performance profile, see Provisioning a worker with real-time capabilities.

12.2. ABOUT DISCOVERY MODE FOR LATENCY TESTS

Use discovery mode to validate the functionality of a cluster without altering its configuration. Existing environment configurations are used for the tests. The tests can 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 preconfigured configuration items is done, and the test environment can be immediately used for another test run.

IMPORTANT

When running the latency tests, **always** run the tests with **-e DISCOVERY_MODE=true** and **-ginkgo.focus** set to the appropriate latency test. If you do not run the latency tests in discovery mode, your existing live cluster performance profile configuration will be modified by the test run.

Limiting the nodes used during tests

The nodes on which the tests are executed can be limited by specifying a **NODES_SELECTOR** environment variable, for example, **-e NODES_SELECTOR=node-role.kubernetes.io/worker-cnf**. Any resources created by the test are limited to nodes with matching labels.

NOTE

If you want to override the default worker pool, pass the **-e ROLE_WORKER_CNF=<custom_worker_pool>** variable to the command specifying an appropriate label.

12.3. MEASURING LATENCY

The **cnf-tests** image uses three tools to measure the latency of the system:

- **hwlatdetect**
- **cyclictest**
- **oslat**

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

hwlatdetect

Measures the baseline that the bare-metal hardware can achieve. Before proceeding with the next latency test, ensure that the latency reported by **hwlatdetect** meets the required threshold because you cannot fix hardware latency spikes by operating system tuning.

cyclictest

Verifies the real-time kernel scheduler latency after **hwlatdetect** passes validation. The **cyclictest** tool schedules a repeated timer and measures the difference between the desired and the actual trigger times. The difference can uncover basic issues with the tuning caused by interrupts or process priorities. The tool must run on a real-time kernel.

oslat

Behaves similarly to a CPU-intensive DPDK application and measures all the interruptions and disruptions to the busy loop that simulates CPU heavy data processing.

The tests introduce the following environment variables:

Table 12.1. Latency test environment variables

Environment variables	Description
LATENCY_TEST_DELAY	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	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	Specifies the amount of time in seconds that the latency test must run. The default value is 300 seconds.
HWLATDETECT_MAXIMUM_LATENCY	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.

Environment variables	Description
CYCLICTEST_MAXIMUM_LATENCY	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	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	Unified variable that specifies the maximum acceptable latency in microseconds. Applicable for all available latency tools.
LATENCY_TEST_RUN	Boolean parameter that indicates whether the tests should run. **LATENCY_TEST_RUN** is set to **false** by default. To run the latency tests, set this value to **true**.

NOTE

Variables that are specific to a latency tool take precedence over unified variables. For example, if **OSLAT_MAXIMUM_LATENCY** is set to 30 microseconds and **MAXIMUM_LATENCY** is set to 10 microseconds, the **oslat** test will run with maximum acceptable latency of 30 microseconds.

12.4. RUNNING THE LATENCY TESTS

Run the cluster latency tests to validate node tuning for your Cloud-native Network Functions (CNF) workload.

IMPORTANT

Always run the latency tests with **DISCOVERY_MODE=true** set. If you don’t, the test suite will make changes to the running cluster configuration.

NOTE

When executing **podman** commands as a non-root or non-privileged user, mounting paths can fail with **permission denied** errors. To make the **podman** command work, append **:Z** to the volumes creation; for example, **-v $(pwd)/:/kubeconfig:Z**. This allows **podman** to do the proper SELinux relabeling.

Procedure

1. Open a shell prompt in the directory containing the **kubeconfig** file.
   You provide the test image with a **kubeconfig** file in current directory and its related **$KUBECONFIG** environment variable, mounted through a volume. This allows the running container to use the **kubeconfig** file from inside the container.

2. Run the latency tests by entering the following command:

```
$ podman run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig \
-e LATENCY_TEST_RUN=true -e DISCOVERY_MODE=true -e FEATURES=performance
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.12 \
/usr/bin/test-run.sh -ginkgo.focus="\[performance\]\ Latency\ Test"
```

3. Optional: Append **-ginkgo.dryRun** to run the latency tests in dry-run mode. This is useful for checking what the tests run.

4. Optional: Append **-ginkgo.v** to run the tests with increased verbosity.

5. Optional: To run the latency tests against a specific performance profile, run the following command, substituting appropriate values:

```
$ podman run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig \
-e LATENCY_TEST_RUN=true -e FEATURES=performance -e
LATENCY_TEST_RUNTIME=600 -e MAXIMUM_LATENCY=20 \
-e PERF_TEST_PROFILE=<performance_profile> registry.redhat.io/openshift4/cnf-tests-
rhel8:v4.12 \
/usr/bin/test-run.sh -ginkgo.focus="[performance]\ Latency\ Test"
```

where:

<performance_profile>
Is the name of the performance profile you want to run the latency tests against.

IMPORTANT

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

12.4.1. Running hwlatdetect

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

IMPORTANT

Always run the latency tests with **DISCOVERY_MODE=true** set. If you don't, the test suite will make changes to the running cluster configuration.

NOTE

When executing **podman** commands as a non-root or non-privileged user, mounting paths can fail with **permission denied** errors. To make the **podman** command work, append **:Z** to the volumes creation; for example, **-v $(pwd)/:/kubeconfig:Z**. This allows **podman** to do the proper SELinux relabeling.

Prerequisites

- You have installed the real-time kernel in the cluster.
- You have logged in to **registry.redhat.io** with your Customer Portal credentials.

Procedure

- To run the **hwlatdetect** tests, run the following command, substituting variable values as appropriate:

```
$ podman run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig \
-e LATENCY_TEST_RUN=true -e DISCOVERY_MODE=true -e FEATURES=performance -
e ROLE_WORKER_CNF=worker-cnf \
-e LATENCY_TEST_RUNTIME=600 -e MAXIMUM_LATENCY=20 \
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.12 \
/usr/bin/test-run.sh -ginkgo.v -ginkgo.focus="hwlatdetect"
```

The **hwlatdetect** test runs for 10 minutes (600 seconds). The test runs successfully when the maximum observed latency is lower than **MAXIMUM_LATENCY** (20 μs).

If the results exceed the latency threshold, the test fails.

IMPORTANT

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

Example failure output

```
running /usr/bin/cnftests -ginkgo.v -ginkgo.focus=hwlatdetect
I0908 15:25:20.023712      27 request.go:601] Waited for 1.046586367s due to client-side
throttling, not priority and fairness, request:
GET:https://api.hlxcl6.lab.eng.tlv2.redhat.com:6443/apis/imageregistry.operator.openshift.io/v1?
timeout=32s
Running Suite: CNF Features e2e integration tests
=============================================
Random Seed: 1662650718
Will run 1 of 194 specs

[...]

• Failure [283.574 seconds]
[performance] Latency Test
/remote-source/app/vendor/github.com/openshift/cluster-node-tuning-
operator/test/e2e/performanceprofile/functests/4_latency/latency.go:62
  with the hwlatdetect image
  /remote-source/app/vendor/github.com/openshift/cluster-node-tuning-
operator/test/e2e/performanceprofile/functests/4_latency/latency.go:228
    should succeed [It]
    /remote-source/app/vendor/github.com/openshift/cluster-node-tuning-
operator/test/e2e/performanceprofile/functests/4_latency/latency.go:236

    Log file created at: 2022/09/08 15:25:27
    Running on machine: hwlatdetect-b6n4n
    Binary: Built with gc go1.17.12 for linux/amd64
    Log line format: [IWEF]mmdd hh:mm:ss.uuuuuu threadid file:line] msg
    I0908 15:25:27.160620       1 node.go:39] Environment information: /proc/cmdline:
BOOT_IMAGE=(hd1,gpt3)/ostree/rhcos-
c6491e1eedf6c1f12ef7b95e14ee720bf48359750ac900b7863c625769ef5fb9/vmlinuz-4.18.0-
372.19.1.el8_6.x86_64 random.trust_cpu=on console=tty0 console=ttyS0,115200n8
ignition.platform.id=metal
ostree=/ostree/boot.1/rhcos/c6491e1eedf6c1f12ef7b95e14ee720bf48359750ac900b7863c625
769ef5fb9/0 ip=dhcp root=UUID=5f80c283-f6e6-4a27-9b47-a287157483b2 rw
```

```
rootflags=prjquota boot=UUID=773bf59a-bafd-48fc-9a87-f62252d739d3 skew_tick=1
nohz=on rcu_nocbs=0-3 tuned.non_isolcpus=0000ffff,ffffffff,fffffff0
systemd.cpu_affinity=4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29
,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,
60,61,62,63,64,65,66,67,68,69,70,71,72,73,74,75,76,77,78,79 intel_iommu=on iommu=pt
isolcpus=managed_irq,0-3 nohz_full=0-3 tsc=nowatchdog nosoftlockup nmi_watchdog=0
mce=off skew_tick=1 rcutree.kthread_prio=11 + +
   I0908 15:25:27.160830       1 node.go:46] Environment information: kernel version 4.18.0-
372.19.1.el8_6.x86_64
   I0908 15:25:27.160857       1 main.go:50] running the hwlatdetect command with
arguments [/usr/bin/hwlatdetect --threshold 1 --hardlimit 1 --duration 100 --window
10000000us --width 950000us]
   F0908 15:27:10.603523       1 main.go:53] failed to run hwlatdetect command; out:
hwlatdetect:  test duration 100 seconds
     detector: tracer
     parameters:
         Latency threshold: 1us 1
         Sample window:     10000000us
         Sample width:      950000us
       Non-sampling period:  9050000us
         Output File:       None

   Starting test
   test finished
   Max Latency: 326us 2
   Samples recorded: 5
   Samples exceeding threshold: 5
   ts: 1662650739.017274507, inner:6, outer:6
   ts: 1662650749.257272414, inner:14, outer:326
   ts: 1662650779.977272835, inner:314, outer:12
   ts: 1662650800.457272384, inner:3, outer:9
   ts: 1662650810.697273520, inner:3, outer:2

[...]

JUnit report was created: /junit.xml/cnftests-junit.xml


Summarizing 1 Failure:

[Fail] [performance] Latency Test with the hwlatdetect image [It] should succeed
/remote-source/app/vendor/github.com/openshift/cluster-node-tuning-
operator/test/e2e/performanceprofile/functests/4_latency/latency.go:476

Ran 1 of 194 Specs in 365.797 seconds
FAIL! -- 0 Passed | 1 Failed | 0 Pending | 193 Skipped
--- FAIL: TestTest (366.08s)
FAIL
```

1 You can configure the latency threshold by using the **MAXIMUM_LATENCY** or the **HWLATDETECT_MAXIMUM_LATENCY** environment variables.

2 The maximum latency value measured during the test.

Example hwlatdetect test results

You can capture the following types of results:

- Rough results that are gathered after each run to create a history of impact on any changes made throughout the test.

- The combined set of the rough tests with the best results and configuration settings.

Example of good results

```
hwlatdetect: test duration 3600 seconds
detector: tracer
parameters:
Latency threshold: 10us
Sample window: 1000000us
Sample width: 950000us
Non-sampling period: 50000us
Output File: None

Starting test
test finished
Max Latency: Below threshold
Samples recorded: 0
```

The **hwlatdetect** tool only provides output if the sample exceeds the specified threshold.

Example of bad results

```
hwlatdetect: test duration 3600 seconds
detector: tracer
parameters:Latency threshold: 10usSample window: 1000000us
Sample width: 950000usNon-sampling period: 50000usOutput File: None

Starting tests:1610542421.275784439, inner:78, outer:81
ts: 1610542444.330561619, inner:27, outer:28
ts: 1610542445.332549975, inner:39, outer:38
ts: 1610542541.568546097, inner:47, outer:32
ts: 1610542590.681548531, inner:13, outer:17
ts: 1610543033.818801482, inner:29, outer:30
ts: 1610543080.938801990, inner:90, outer:76
ts: 1610543129.065549639, inner:28, outer:39
ts: 1610543474.859552115, inner:28, outer:35
ts: 1610543523.973856571, inner:52, outer:49
ts: 1610543572.089799738, inner:27, outer:30
ts: 1610543573.091550771, inner:34, outer:28
ts: 1610543574.093555202, inner:116, outer:63
```

The output of **hwlatdetect** shows that multiple samples exceed the threshold. However, the same output can indicate different results based on the following factors:

- The duration of the test

- The number of CPU cores

- The host firmware settings

WARNING

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

Not all latency spikes are hardware related. Ensure that you tune the host firmware to meet your workload requirements. For more information, see Setting firmware parameters for system tuning.

12.4.2. Running cyclictest

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

IMPORTANT

Always run the latency tests with **DISCOVERY_MODE=true** set. If you don't, the test suite will make changes to the running cluster configuration.

NOTE

When executing **podman** commands as a non-root or non-privileged user, mounting paths can fail with **permission denied** errors. To make the **podman** command work, append **:Z** to the volumes creation; for example, **-v $(pwd)/:/kubeconfig:Z**. This allows **podman** to do the proper SELinux relabeling.

Prerequisites

- You have logged in to **registry.redhat.io** with your Customer Portal credentials.
- You have installed the real-time kernel in the cluster.
- You have applied a cluster performance profile by using Node Tuning Operator.

Procedure

- To perform the **cyclictest**, run the following command, substituting variable values as appropriate:

```
$ podman run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig \
-e LATENCY_TEST_RUN=true -e DISCOVERY_MODE=true -e FEATURES=performance -
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:v4.12 \
/usr/bin/test-run.sh -ginkgo.v -ginkgo.focus="cyclictest"
```

The command runs the **cyclictest** tool for 10 minutes (600 seconds). The test runs successfully when the maximum observed latency is lower than **MAXIMUM_LATENCY** (in this example, 20 μs). Latency spikes of 20 μs and above are generally not acceptable for telco RAN workloads.

If the results exceed the latency threshold, the test fails.

IMPORTANT

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

Example failure output

```
running /usr/bin/cnftests -ginkgo.v -ginkgo.focus=cyclictest
I0908 13:01:59.193776      27 request.go:601] Waited for 1.046228824s due to client-side
throttling, not priority and fairness, request: GET:https://api.compute-
1.example.com:6443/apis/packages.operators.coreos.com/v1?timeout=32s
Running Suite: CNF Features e2e integration tests
=================================================
Random Seed: 1662642118
Will run 1 of 194 specs

[...]

Summarizing 1 Failure:

[Fail] [performance] Latency Test with the cyclictest image [It] should succeed
/remote-source/app/vendor/github.com/openshift/cluster-node-tuning-
operator/test/e2e/performanceprofile/functests/4_latency/latency.go:220

Ran 1 of 194 Specs in 161.151 seconds
FAIL! -- 0 Passed | 1 Failed | 0 Pending | 193 Skipped
--- FAIL: TestTest (161.48s)
FAIL
```

Example cyclictest results

The same output can indicate different results for different workloads. For example, spikes up to 18μs are acceptable for 4G DU workloads, but not for 5G DU workloads.

Example of good results

```
running cmd: cyclictest -q -D 10m -p 1 -t 16 -a 2,4,6,8,10,12,14,16,54,56,58,60,62,64,66,68 -h 30 -i
1000 -m
# Histogram
000000 000000   000000  000000  000000  000000  000000  000000  000000  000000  000000
000000  000000  000000  000000  000000  000000
000001 000000   000000  000000  000000  000000  000000  000000  000000  000000  000000
000000  000000  000000  000000  000000  000000
000002 579506   535967  418614  573648  532870  529897  489306  558076  582350  585188
583793  223781  532480  569130  472250  576043
More histogram entries ...
# Total: 000600000 000600000 000600000 000599999 000599999 000599999 000599998
000599998 000599998 000599997 000599997 000599996 000599996 000599995 000599995
000599995
# Min Latencies: 00002 00002 00002 00002 00002 00002 00002 00002 00002 00002 00002 00002
00002 00002 00002 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: cyclictest -q -D 10m -p 1 -t 16 -a 2,4,6,8,10,12,14,16,54,56,58,60,62,64,66,68 -h 30 -i
1000 -m
# Histogram
000000 000000   000000  000000  000000  000000  000000  000000  000000  000000  000000
000000  000000  000000  000000  000000  000000
000001 000000   000000  000000  000000  000000  000000  000000  000000  000000  000000
000000  000000  000000  000000  000000  000000
000002 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
```

12.4.3. Running oslat

The **oslat** test simulates a CPU-intensive DPDK application and measures all the interruptions and disruptions to test how the cluster handles CPU heavy data processing.

IMPORTANT

Always run the latency tests with **DISCOVERY_MODE=true** set. If you don’t, the test suite will make changes to the running cluster configuration.

NOTE

When executing **podman** commands as a non-root or non-privileged user, mounting paths can fail with **permission denied** errors. To make the **podman** command work, append **:Z** to the volumes creation; for example, **-v $(pwd)/:/kubeconfig:Z**. This allows **podman** to do the proper SELinux relabeling.

Prerequisites

- You have logged in to **registry.redhat.io** with your Customer Portal credentials.
- You have applied a cluster performance profile by using the Node Tuning Operator.

Procedure

- To perform the **oslat** test, run the following command, substituting variable values as appropriate:

```
$ podman run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig \
-e LATENCY_TEST_RUN=true -e DISCOVERY_MODE=true -e FEATURES=performance -
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:v4.12 \
/usr/bin/test-run.sh -ginkgo.v -ginkgo.focus="oslat"
```

LATENCY_TEST_CPUS specifies the list of CPUs to test with the **oslat** command.

The command runs the **oslat** tool for 10 minutes (600 seconds). The test runs successfully when the maximum observed latency is lower than **MAXIMUM_LATENCY** (20 μs).

If the results exceed the latency threshold, the test fails.

IMPORTANT

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

Example failure output

```
running /usr/bin/cnftests -ginkgo.v -ginkgo.focus=oslat
I0908 12:51:55.999393      27 request.go:601] Waited for 1.044848101s due to client-side
throttling, not priority and fairness, request: GET:https://compute-
1.example.com:6443/apis/machineconfiguration.openshift.io/v1?timeout=32s
Running Suite: CNF Features e2e integration tests
=====================================================
Random Seed: 1662641514
Will run 1 of 194 specs

[...]

• Failure [77.833 seconds]
[performance] Latency Test
/remote-source/app/vendor/github.com/openshift/cluster-node-tuning-
operator/test/e2e/performanceprofile/functests/4_latency/latency.go:62
  with the oslat image
  /remote-source/app/vendor/github.com/openshift/cluster-node-tuning-
operator/test/e2e/performanceprofile/functests/4_latency/latency.go:128
    should succeed [It]
    /remote-source/app/vendor/github.com/openshift/cluster-node-tuning-
operator/test/e2e/performanceprofile/functests/4_latency/latency.go:153

    The current latency 304 is bigger than the expected one 1 : 1

[...]

Summarizing 1 Failure:

[Fail] [performance] Latency Test with the oslat image [It] should succeed
/remote-source/app/vendor/github.com/openshift/cluster-node-tuning-
operator/test/e2e/performanceprofile/functests/4_latency/latency.go:177

Ran 1 of 194 Specs in 161.091 seconds
FAIL! -- 0 Passed | 1 Failed | 0 Pending | 193 Skipped
--- FAIL: TestTest (161.42s)
FAIL
```

1 In this example, the measured latency is outside the maximum allowed value.

12.5. GENERATING A LATENCY TEST FAILURE REPORT

Use the following procedures to generate a JUnit latency test output and test failure report.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have logged in as a user with **cluster-admin** privileges.

Procedure

- Create a test failure report with information about the cluster state and resources for troubleshooting by passing the **--report** parameter with the path to where the report is dumped:

```
$ podman run -v $(pwd)/:/kubeconfig:Z -v $(pwd)/reportdest:<report_folder_path> \
-e KUBECONFIG=/kubeconfig/kubeconfig  -e DISCOVERY_MODE=true -e
FEATURES=performance \
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.12 \
/usr/bin/test-run.sh --report <report_folder_path> \
-ginkgo.focus="\[performance\]\ Latency\ Test"
```

where:

<report_folder_path>

Is the path to the folder where the report is generated.

12.6. GENERATING A JUNIT LATENCY TEST REPORT

Use the following procedures to generate a JUnit latency test output and test failure report.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have logged in as a user with **cluster-admin** privileges.

Procedure

- Create a JUnit-compliant XML report by passing the **--junit** parameter together with the path to where the report is dumped:

```
$ podman run -v $(pwd)/:/kubeconfig:Z -v $(pwd)/junitdest:<junit_folder_path> \
-e KUBECONFIG=/kubeconfig/kubeconfig -e DISCOVERY_MODE=true -e
FEATURES=performance \
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.12 \
/usr/bin/test-run.sh --junit <junit_folder_path> \
-ginkgo.focus="\[performance\]\ Latency\ Test"
```

where:

<junit_folder_path>

Is the path to the folder where the junit report is generated

12.7. RUNNING LATENCY TESTS ON A SINGLE-NODE OPENSHIFT CLUSTER

You can run latency tests on single-node OpenShift clusters.

IMPORTANT

Always run the latency tests with **DISCOVERY_MODE=true** set. If you don't, the test suite will make changes to the running cluster configuration.

NOTE

When executing **podman** commands as a non-root or non-privileged user, mounting paths can fail with **permission denied** errors. To make the **podman** command work, append **:Z** to the volumes creation; for example, **-v $(pwd)/:/kubeconfig:Z**. This allows **podman** to do the proper SELinux relabeling.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have logged in as a user with **cluster-admin** privileges.

Procedure

- To run the latency tests on a single-node OpenShift cluster, run the following command:

```
$ podman run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig \
-e DISCOVERY_MODE=true -e FEATURES=performance -e
ROLE_WORKER_CNF=master \
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.12 \
/usr/bin/test-run.sh -ginkgo.focus="\[performance\]\ Latency\ Test"
```

NOTE

ROLE_WORKER_CNF=master is required because master is the only machine pool to which the node belongs. For more information about setting the required **MachineConfigPool** for the latency tests, see "Prerequisites for running latency tests".

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

12.8. RUNNING LATENCY TESTS IN A DISCONNECTED CLUSTER

The CNF tests image can run tests in a disconnected cluster that is not able to reach external registries. This requires two steps:

1. Mirroring the **cnf-tests** image to the custom disconnected registry.
2. Instructing the tests to consume the images from the custom disconnected registry.

Mirroring the images to a custom registry accessible from the cluster

A **mirror** executable is shipped in the image to provide the input required by **oc** to mirror the test image to a local registry.

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

```
$ podman run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig \
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.12 \
/usr/bin/mirror -registry <disconnected_registry> | oc image mirror -f -
```

where:

<disconnected_registry>

Is the disconnected mirror registry you have configured, for example, **my.local.registry:5000/**.

2. When you have mirrored the **cnf-tests** image into the disconnected registry, you must override the original registry used to fetch the images when running the tests, for example:

```
$ podman run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig \
-e DISCOVERY_MODE=true -e FEATURES=performance -e IMAGE_REGISTRY="
<disconnected_registry>" \
-e CNF_TESTS_IMAGE="cnf-tests-rhel8:v4.12" \
/usr/bin/test-run.sh -ginkgo.focus="\[performance\]\ Latency\ Test"
```

Configuring the tests to consume images from a custom registry

You can run the latency tests using a custom test image and image registry using **CNF_TESTS_IMAGE** and **IMAGE_REGISTRY** variables.

- To configure the latency tests to use a custom test image and image registry, run the following command:

```
$ podman run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig \
-e IMAGE_REGISTRY="<custom_image_registry>" \
-e CNF_TESTS_IMAGE="<custom_cnf-tests_image>" \
-e FEATURES=performance \
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.12 /usr/bin/test-run.sh
```

where:

<custom_image_registry>

is the custom image registry, for example, **custom.registry:5000/**.

<custom_cnf-tests_image>

is the custom cnf-tests image, for example, **custom-cnf-tests-image:latest**.

Mirroring images to the cluster OpenShift image registry

OpenShift Container Platform provides a built-in container image registry, which runs as a standard workload on the cluster.

Procedure

1. Gain external access to the registry by exposing it with a route:

```
$ oc patch configs.imageregistry.operator.openshift.io/cluster --patch '{"spec":
{"defaultRoute":true}}' --type=merge
```

2. Fetch the registry endpoint by running the following command:

```
$ REGISTRY=$(oc get route default-route -n openshift-image-registry --template='{{
.spec.host }}')
```

3. Create a namespace for exposing the images:

```
$ oc create ns cnftests
```

4. Make the image stream available to all the namespaces used for tests. This is required to allow the tests namespaces to fetch the images from the **cnf-tests** image stream. Run the following commands:

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

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

5. Retrieve the docker secret name and auth token by running the following commands:

   ```
   $ SECRET=$(oc -n cnftests get secret | grep builder-docker | awk {'print $1'}
   ```

   ```
   $ TOKEN=$(oc -n cnftests get secret $SECRET -o jsonpath="{.data['\.dockercfg']}" | base64 --decode | jq '.["image-registry.openshift-image-registry.svc:5000"].auth')
   ```

6. Create a **dockerauth.json** file, for example:

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

7. Do the image mirroring:

   ```
   $ podman run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig \
   registry.redhat.io/openshift4/cnf-tests-rhel8:4.12 \
   /usr/bin/mirror -registry $REGISTRY/cnftests |  oc image mirror --insecure=true \
   -a=$(pwd)/dockerauth.json -f -
   ```

8. Run the tests:

   ```
   $ podman run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig \
   -e DISCOVERY_MODE=true -e FEATURES=performance -e IMAGE_REGISTRY=image-registry.openshift-image-registry.svc:5000/cnftests \
   cnf-tests-local:latest /usr/bin/test-run.sh -ginkgo.focus="\[performance\]\ Latency\ Test"
   ```

Mirroring a different set of test images

You can optionally change the default upstream images that are mirrored for the latency tests.

Procedure

1. The **mirror** command tries to mirror the upstream images by default. This can be overridden by passing a file with the following format to the image:

   ```
   [
       {
           "registry": "public.registry.io:5000",
           "image": "imageforcnftests:4.12"
       }
   ]
   ```

2. Pass the file to the **mirror** command, for example saving it locally as **images.json**. With the following command, the local path is mounted in **/kubeconfig** inside the container and that can be passed to the mirror command.

```
$ podman run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig \
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.12 /usr/bin/mirror \
--registry "my.local.registry:5000/" --images "/kubeconfig/images.json" \
|  oc image mirror -f -
```

12.9. TROUBLESHOOTING ERRORS WITH THE CNF-TESTS CONTAINER

To run latency tests, the cluster must be accessible from within the **cnf-tests** container.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have logged in as a user with **cluster-admin** privileges.

Procedure

- Verify that the cluster is accessible from inside the **cnf-tests** container by running the following command:

```
$ podman run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig \
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.12 \
oc get nodes
```

If this command does not work, an error related to spanning across DNS, MTU size, or firewall access might be occurring.

CHAPTER 13. IMPROVING CLUSTER STABILITY IN HIGH LATENCY ENVIRONMENTS USING WORKER LATENCY PROFILES

If the cluster administrator has performed latency tests for platform verification, they can discover the need to adjust the operation of the cluster to ensure stability in cases of high latency. The cluster administrator need change only one parameter, recorded in a file, which controls four parameters affecting how supervisory processes read status and interpret the health of the cluster. Changing only the one parameter provides cluster tuning in an easy, supportable manner.

The **Kubelet** process provides the starting point for monitoring cluster health. The **Kubelet** sets status values for all nodes in the OpenShift Container Platform cluster. The Kubernetes Controller Manager (**kube controller**) reads the status values every 10 seconds, by default. If the **kube controller** cannot read a node status value, it loses contact with that node after a configured period. The default behavior is:

1. The node controller on the control plane updates the node health to **Unhealthy** and marks the node **Ready** condition`Unknown`.

2. In response, the scheduler stops scheduling pods to that node.

3. The Node Lifecycle Controller adds a **node.kubernetes.io/unreachable** taint with a **NoExecute** effect to the node and schedules any pods on the node for eviction after five minutes, by default.

This behavior can cause problems if your network is prone to latency issues, especially if you have nodes at the network edge. In some cases, the Kubernetes Controller Manager might not receive an update from a healthy node due to network latency. The **Kubelet** evicts pods from the node even though the node is healthy.

To avoid this problem, you can use *worker latency profiles* to adjust the frequency that the **Kubelet** and the Kubernetes Controller Manager wait for status updates before taking action. These adjustments help to ensure that your cluster runs properly if network latency between the control plane and the worker nodes is not optimal.

These worker latency profiles contain three sets of parameters that are pre-defined with carefully tuned values to control the reaction of the cluster to increased latency. No need to experimentally find the best values manually.

You can configure worker latency profiles when installing a cluster or at any time you notice increased latency in your cluster network.

13.1. UNDERSTANDING WORKER LATENCY PROFILES

Worker latency profiles are four different categories of carefully-tuned parameters. The four parameters which implement these values are **node-status-update-frequency**, **node-monitor-grace-period**, **default-not-ready-toleration-seconds** and **default-unreachable-toleration-seconds**. These parameters can use values which allow you control the reaction of the cluster to latency issues without needing to determine the best values using manual methods.

IMPORTANT

Setting these parameters manually is not supported. Incorrect parameter settings adversely affect cluster stability.

All worker latency profiles configure the following parameters:

node-status-update-frequency
: Specifies how often the kubelet posts node status to the API server.

node-monitor-grace-period
: Specifies the amount of time in seconds that the Kubernetes Controller Manager waits for an update from a kubelet before marking the node unhealthy and adding the **node.kubernetes.io/not-ready** or **node.kubernetes.io/unreachable** taint to the node.

default-not-ready-toleration-seconds
: Specifies the amount of time in seconds after marking a node unhealthy that the Kube API Server Operator waits before evicting pods from that node.

default-unreachable-toleration-seconds
: Specifies the amount of time in seconds after marking a node unreachable that the Kube API Server Operator waits before evicting pods from that node.

The following Operators monitor the changes to the worker latency profiles and respond accordingly:

- The Machine Config Operator (MCO) updates the **node-status-update-frequency** parameter on the worker nodes.
- The Kubernetes Controller Manager updates the **node-monitor-grace-period** parameter on the control plane nodes.
- The Kubernetes API Server Operator updates the **default-not-ready-toleration-seconds** and **default-unreachable-toleration-seconds** parameters on the control plane nodes.

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** updates it’s status every 10 seconds (**node-status-update-frequency**). The **Kube Controller Manager** checks the statuses of **Kubelet** every 5 seconds (**node-monitor-grace-period**).
The Kubernetes Controller Manager waits 40 seconds for a status update from **Kubelet** before considering the **Kubelet** unhealthy. If no status is made available to the Kubernetes Controller Manager, it then marks the node with the **node.kubernetes.io/not-ready** or **node.kubernetes.io/unreachable** taint and evicts the pods on that node.

If a pod on that node has the **NoExecute** taint, the pod is run according to **tolerationSeconds**. If the pod has no taint, it will be evicted in 300 seconds (**default-not-ready-toleration-seconds** and **default-unreachable-toleration-seconds** settings of the **Kube API Server**).

Profile	Component	Parameter	Value
Default	kubelet	**node-status-update-frequency**	10s
	Kubelet Controller Manager	**node-monitor-grace-period**	40s

Profile	Component	Parameter	Value
	Kubernetes API Server Operator	**default-not-ready-toleration-seconds**	300s
	Kubernetes API Server Operator	**default-unreachable-toleration-seconds**	300s

Medium worker latency profile

Use the **MediumUpdateAverageReaction** profile if the network latency is slightly higher than usual. The **MediumUpdateAverageReaction** profile reduces the frequency of kubelet updates to 20 seconds and changes the period that the Kubernetes Controller Manager waits for those updates to 2 minutes. The pod eviction period for a pod on that node is reduced to 60 seconds. If the pod has the **tolerationSeconds** parameter, the eviction waits for the period specified by that parameter.

The Kubernetes Controller Manager waits for 2 minutes to consider a node unhealthy. In another minute, the eviction process starts.

Profile	Component	Parameter	Value
MediumUpdateAverageReaction	kubelet	**node-status-update-frequency**	20s
	Kubelet Controller Manager	**node-monitor-grace-period**	2m
	Kubernetes API Server Operator	**default-not-ready-toleration-seconds**	60s
	Kubernetes API Server Operator	**default-unreachable-toleration-seconds**	60s

Low worker latency profile

Use the **LowUpdateSlowReaction** profile if the network latency is extremely high. The **LowUpdateSlowReaction** profile reduces the frequency of kubelet updates to 1 minute and changes the period that the Kubernetes Controller Manager waits for those updates to 5 minutes. The pod eviction period for a pod on that node is reduced to 60 seconds. If the pod has the **tolerationSeconds** parameter, the eviction waits for the period specified by that parameter.

The Kubernetes Controller Manager waits for 5 minutes to consider a node unhealthy. In another minute, the eviction process starts.

Profile	Component	Parameter	Value
LowUpdateSlowReaction	kubelet	**node-status-update-frequency**	1m
	Kubelet Controller Manager	**node-monitor-grace-period**	5m
	Kubernetes API Server Operator	**default-not-ready-toleration-seconds**	60s
	Kubernetes API Server Operator	**default-unreachable-toleration-seconds**	60s

13.2. IMPLEMENTING WORKER LATENCY PROFILES AT CLUSTER CREATION

IMPORTANT

To edit the configuration of the installer, you will first need to use the command **openshift-install create manifests** to create the default node manifest as well as other manifest YAML files. This file structure must exist before we can add workerLatencyProfile. The platform on which you are installing may have varying requirements. Refer to the Installing section of the documentation for your specific platform.

The **workerLatencyProfile** must be added to the manifest in the following sequence:

1. Create the manifest needed to build the cluster, using a folder name appropriate for your installation.

2. Create a YAML file to define **config.node**. The file must be in the **manifests** directory.

3. When defining **workerLatencyProfile** in the manifest for the first time, specify any of the profiles at cluster creation time: **Default**, **MediumUpdateAverageReaction** or **LowUpdateSlowReaction**.

Verification

- Here is an example manifest creation showing the **spec.workerLatencyProfile Default** value in the manifest file:

  ```
  $ openshift-install create manifests --dir=<cluster-install-dir>
  ```

- Edit the manifest and add the value. In this example we use **vi** to show an example manifest file with the "Default" **workerLatencyProfile** value added:

```
$ vi <cluster-install-dir>/manifests/config-node-default-profile.yaml
```

Example output

```
apiVersion: config.openshift.io/v1
kind: Node
metadata:
name: cluster
spec:
workerLatencyProfile: "Default"
```

13.3. USING AND CHANGING WORKER LATENCY PROFILES

To change a worker latency profile to deal with network latency, edit the **node.config** object to add the name of the profile. You can change the profile at any time as latency increases or decreases.

You must move one worker latency profile at a time. For example, you cannot move directly from the **Default** profile to the **LowUpdateSlowReaction** worker latency profile. You must move from the **Default** worker latency profile to the **MediumUpdateAverageReaction** profile first, then to **LowUpdateSlowReaction**. Similarly, when returning to the **Default** profile, you must move from the low profile to the medium profile first, then to **Default**.

> **NOTE**
>
> You can also configure worker latency profiles upon installing an OpenShift Container Platform cluster.

Procedure

To move from the default worker latency profile:

1. Move to the medium worker latency profile:

   a. Edit the **node.config** object:

   ```
   $ oc edit nodes.config/cluster
   ```

   b. Add **spec.workerLatencyProfile: MediumUpdateAverageReaction**:

   Example node.config object

   ```
   apiVersion: config.openshift.io/v1
   kind: Node
   metadata:
     annotations:
       include.release.openshift.io/ibm-cloud-managed: "true"
       include.release.openshift.io/self-managed-high-availability: "true"
       include.release.openshift.io/single-node-developer: "true"
       release.openshift.io/create-only: "true"
     creationTimestamp: "2022-07-08T16:02:51Z"
     generation: 1
     name: cluster
     ownerReferences:
   ```

```
  - apiVersion: config.openshift.io/v1
    kind: ClusterVersion
    name: version
    uid: 36282574-bf9f-409e-a6cd-3032939293eb
  resourceVersion: "1865"
  uid: 0c0f7a4c-4307-4187-b591-6155695ac85b
spec:
  workerLatencyProfile: MediumUpdateAverageReaction 1

# ...
```

1 Specifies the medium worker latency policy.

Scheduling on each worker node is disabled as the change is being applied.

2. Optional: Move to the low worker latency profile:

   a. Edit the **node.config** object:

   ```
   $ oc edit nodes.config/cluster
   ```

   b. Change the **spec.workerLatencyProfile** value to **LowUpdateSlowReaction**:

   Example node.config object

   ```
   apiVersion: config.openshift.io/v1
   kind: Node
   metadata:
     annotations:
       include.release.openshift.io/ibm-cloud-managed: "true"
       include.release.openshift.io/self-managed-high-availability: "true"
       include.release.openshift.io/single-node-developer: "true"
       release.openshift.io/create-only: "true"
     creationTimestamp: "2022-07-08T16:02:51Z"
     generation: 1
     name: cluster
     ownerReferences:
     - apiVersion: config.openshift.io/v1
       kind: ClusterVersion
       name: version
       uid: 36282574-bf9f-409e-a6cd-3032939293eb
     resourceVersion: "1865"
     uid: 0c0f7a4c-4307-4187-b591-6155695ac85b
   spec:
     workerLatencyProfile: LowUpdateSlowReaction 1

   # ...
   ```

   1 Specifies use of the low worker latency policy.

Scheduling on each worker node is disabled as the change is being applied.

Verification

- When all nodes return to the **Ready** condition, you can use the following command to look in the Kubernetes Controller Manager to ensure it was applied:

  ```
  $ oc get KubeControllerManager -o yaml | grep -i workerlatency -A 5 -B 5
  ```

  Example output

  ```
  # ...
    - lastTransitionTime: "2022-07-11T19:47:10Z"
      reason: ProfileUpdated
      status: "False"
      type: WorkerLatencyProfileProgressing
    - lastTransitionTime: "2022-07-11T19:47:10Z" 1
      message: all static pod revision(s) have updated latency profile
      reason: ProfileUpdated
      status: "True"
      type: WorkerLatencyProfileComplete
    - lastTransitionTime: "2022-07-11T19:20:11Z"
      reason: AsExpected
      status: "False"
      type: WorkerLatencyProfileDegraded
    - lastTransitionTime: "2022-07-11T19:20:36Z"
      status: "False"
  # ...
  ```

  1 Specifies that the profile is applied and active.

To change the medium profile to default or change the default to medium, edit the **node.config** object and set the **spec.workerLatencyProfile** parameter to the appropriate value.

13.4. EXAMPLE STEPS FOR DISPLAYING RESULTING VALUES OF WORKERLATENCYPROFILE

You can display the values in the **workerLatencyProfile** with the following commands.

Verification

1. Check the **default-not-ready-toleration-seconds** and **default-unreachable-toleration-seconds** fields output by the Kube API Server:

   ```
   $ oc get KubeAPIServer -o yaml | grep -A 1 default-
   ```

   Example output

   ```
   default-not-ready-toleration-seconds:
   - "300"
   default-unreachable-toleration-seconds:
   - "300"
   ```

2. Check the values of the **node-monitor-grace-period** field from the Kube Controller Manager:

   ```
   $ oc get KubeControllerManager -o yaml | grep -A 1 node-monitor
   ```

–

Example output

```
node-monitor-grace-period:
- 40s
```

3. Check the **nodeStatusUpdateFrequency** value from the Kubelet. Set the directory **/host** as the root directory within the debug shell. By changing the root directory to **/host**, you can run binaries contained in the host’s executable paths:

```
$ oc debug node/<worker-node-name>
$ chroot /host
# cat /etc/kubernetes/kubelet.conf|grep nodeStatusUpdateFrequency
```

Example output

```
  “nodeStatusUpdateFrequency”: “10s”
```

These outputs validate the set of timing variables for the Worker Latency Profile.

CHAPTER 14. CREATING A PERFORMANCE PROFILE

Learn about the Performance Profile Creator (PPC) and how you can use it to create a performance profile.

NOTE

Currently, disabling CPU load balancing is not supported by cgroup v2. As a result, you might not get the desired behavior from performance profiles if you have cgroup v2 enabled. Enabling cgroup v2 is not recommended if you are using performance profiles.

14.1. 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

14.1.1. Gathering data about your cluster using the must-gather command

The Performance Profile Creator (PPC) tool requires **must-gather** data. As a cluster administrator, run the **must-gather** command to capture information about your cluster.

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 and later, this functionality is 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. Optional: 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
```

2. If a matching label does not exist add a label for a machine config pool (MCP) that matches with the MCP name:

```
$ oc label mcp <mcp_name> <mcp_name>=""
```

3. Navigate to the directory where you want to store the **must-gather** data.

4. Run **must-gather** on your cluster:

```
$ oc adm must-gather --image=<PAO_must_gather_image> --dest-dir=<dir>
```

NOTE

The **must-gather** command 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:v4.12 --dest-dir=<path_to_must-gather>/must-gather
```

5. Create a compressed file from the **must-gather** directory:

```
$ tar cvaf must-gather.tar.gz must-gather/
```

14.1.2. 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: <username>
Password: <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:v4.12 -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
      --per-pod-power-management          Enable Per Pod Power Management
      --power-consumption-mode string     The power consumption mode.  [Valid values:
default, low-latency, ultra-low-latency] (default "default")
      --profile-name string               Name of the performance profile to be created (default
"performance")
      --reserved-cpu-count int            Number of reserved CPUs (required)
      --rt-kernel                         Enable Real Time Kernel (required)
      --split-reserved-cpus-across-numa   Split the Reserved CPUs across NUMA nodes
      --topology-manager-policy string    Kubelet Topology Manager Policy of the performance
profile to be created. [Valid values: single-numa-node, best-effort, restricted] (default
"restricted")
      --user-level-networking             Run with User level Networking(DPDK) enabled
```

4. 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:v4.12 --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:v4.12 --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
```

14.1.2.1. 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:v4.12 --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.

14.1.3. 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:v4.12"
MG_TARBALL=""
DATA_DIR=""

usage() {
  print "Wrapper usage:"
  print "  ${CURRENT_SCRIPT} [-h] [-p image][-t path] -- [performance-profile-creator flags]"
  print ""
  print "Options:"
  print "   -h                  help for ${CURRENT_SCRIPT}"
  print "   -p                  Node Tuning Operator image"
  print "   -t                  path to a must-gather tarball"

  ${IMG_EXISTS_CMD} "${NTO_IMG}" && ${CMD} "${NTO_IMG}" -h
}

function cleanup {
  [ -d "${DATA_DIR}" ] && rm -rf "${DATA_DIR}"
}
trap cleanup EXIT

exit_error() {
  print "error: $*"
  usage
  exit 1
}

print() {
  echo  "$*" >&2
}

check_requirements() {
  ${IMG_EXISTS_CMD} "${NTO_IMG}" || ${IMG_PULL_CMD} "${NTO_IMG}" || \
      exit_error "Node Tuning Operator image not found"

  [ -n "${MG_TARBALL}" ] || exit_error "Must-gather tarball file path is mandatory"
  [ -f "${MG_TARBALL}" ] || exit_error "Must-gather tarball file not found"

  DATA_DIR=$(mktemp -d -t "${CURRENT_SCRIPT}XXXX") || exit_error "Cannot create the
data directory"
  tar -zxf "${MG_TARBALL}" --directory "${DATA_DIR}" || exit_error "Cannot decompress the
must-gather tarball"
  chmod a+rx "${DATA_DIR}"

  return 0
}

main() {
  while getopts ':hp:t:' OPT; do
    case "${OPT}" in
      h)
        usage
        exit 0
        ;;
      p)
        NTO_IMG="${OPTARG}"
        ;;
      t)
        MG_TARBALL="${OPTARG}"
        ;;
      ?)
        exit_error "invalid argument: ${OPTARG}"
      ;;
    esac
  done
  shift $((OPTIND - 1))

  check_requirements || exit 1

  ${CMD} -v "${DATA_DIR}:${MUST_GATHER_VOL}:z" "${NTO_IMG}" "$@" --must-gather-dir-path "${MUST_GATHER_VOL}"
  echo "" 1>&2
}

main "$@"
```

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
      --per-pod-power-management          Enable Per Pod Power Management
      --power-consumption-mode string     The power consumption mode.  [Valid values:
default, low-latency, ultra-low-latency] (default "default")
      --profile-name string               Name of the performance profile to be created (default
"performance")
      --reserved-cpu-count int            Number of reserved CPUs (required)
      --rt-kernel                         Enable Real Time Kernel (required)
      --split-reserved-cpus-across-numa   Split the Reserved CPUs across NUMA nodes
      --topology-manager-policy string    Kubelet Topology Manager Policy of the performance
```

```
profile to be created. [Valid values: single-numa-node, best-effort, restricted] (default
"restricted")
      --user-level-networking           Run with User level Networking(DPDK) enabled
```

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:v4.12**.

2 **-t** is a required wrapper script argument and specifies the path to a **must-gather** tarball.

5. 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.

6. 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
```

7. 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
```

14.1.4. Performance Profile Creator arguments

Table 14.1. Performance Profile Creator arguments

Argument	Description
disable-ht	Disable hyperthreading. Possible values: **true** or **false**. Default: **false**. WARNING 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.
info	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. Possible values: ● **log** ● **JSON** NOTE These options define the output format with the JSON format being reserved for debugging. Default: **log**.
mcp-name	MCP name for example **worker-cnf** corresponding to the target machines. This parameter is required.
must-gather-dir-path	Must gather directory path. This parameter is required. 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.

Argument	Description
offlined-cpu-count	Number of offlined CPUs. NOTE This must be a natural number greater than 0. If not enough logical processors are offlined then error messages are logged. The messages are: 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] Error: failed to compute the reserved and isolated CPUs: please specify the offlined CPU count in the range [0,1]
power-consumption-mode	The power consumption mode. Possible values: - **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. Default: **default**.
per-pod-power-management	Enable per pod power management. You cannot use this argument if you configured **ultra-low-latency** as the power consumption mode. Possible values: **true** or **false**. Default: **false**.
profile-name	Name of the performance profile to create. Default: **performance**.

Argument	Description
reserved-cpu-count	Number of reserved CPUs. This parameter is required. NOTE This must be a natural number. A value of 0 is not allowed.
rt-kernel	Enable real-time kernel. This parameter is required. Possible values: **true** or **false**.
split-reserved-cpus-across-numa	Split the reserved CPUs across NUMA nodes. Possible values: **true** or **false**. Default: **false**.
topology-manager-policy	Kubelet Topology Manager policy of the performance profile to be created. Possible values: ● **single-numa-node** ● **best-effort** ● **restricted** Default: **restricted**.
user-level-networking	Run with user level networking (DPDK) enabled. Possible values: **true** or **false**. Default: **false**.

14.2. REFERENCE PERFORMANCE PROFILES

14.2.1. 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.

14.3. ADDITIONAL RESOURCES

- For more information about the **must-gather** tool, see Gathering data about your cluster .

CHAPTER 15. WORKLOAD PARTITIONING IN SINGLE-NODE OPENSHIFT

In resource-constrained environments, such as single-node OpenShift deployments, use workload partitioning to isolate OpenShift Container Platform services, cluster management workloads, and infrastructure pods to run on a reserved set of CPUs.

The minimum number of reserved CPUs required for the cluster management in single-node OpenShift is four CPU Hyper-Threads (HTs). With workload partitioning, you annotate the set of cluster management pods and a set of typical add-on Operators for inclusion in the cluster management workload partition. These pods operate normally within the minimum size CPU configuration. Additional Operators or workloads outside of the set of minimum cluster management pods require additional CPUs to be added to the workload partition.

Workload partitioning isolates user workloads from platform workloads using standard Kubernetes scheduling capabilities.

The following is an overview of the configurations required for workload partitioning:

- Workload partitioning that uses **/etc/crio/crio.conf.d/01-workload-partitioning** pins the OpenShift Container Platform infrastructure pods to a defined **cpuset** configuration.
- The performance profile pins cluster services such as systemd and kubelet to the CPUs that are defined in the **spec.cpu.reserved** field.

  NOTE

  Using the Node Tuning Operator, you can configure the performance profile to also pin system-level apps for a complete workload partitioning configuration on the node.

- The CPUs that you specify in the performance profile **spec.cpu.reserved** field and the workload partitioning **cpuset** field must match.

Workload partitioning introduces an extended **<workload-type>.workload.openshift.io/cores** resource for each defined CPU pool, or *workload type*. Kubelet advertises the resources and CPU requests by pods allocated to the pool within the corresponding 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.

Additional resources

- For the recommended workload partitioning configuration for single-node OpenShift clusters, see Workload partitioning.

CHAPTER 16. 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 Technology Preview Features Support Scope .

16.1. 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.

16.2. 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.

16.2.1. Installing the Node Observability Operator using the CLI

You can install the Node Observability Operator by using the OpenShift CLI (oc).

Prerequisites

- You have installed the OpenShift CLI (oc).
- You have access to the cluster with **cluster-admin** privileges.

Procedure

1. Confirm that the Node Observability Operator is available by running the following command:

```
$ oc get packagemanifests -n openshift-marketplace node-observability-operator
```

Example output

```
NAME                             CATALOG              AGE
node-observability-operator      Red Hat Operators      9h
```

2. Create the **node-observability-operator** namespace by running the following command:

   ```
   $ oc new-project node-observability-operator
   ```

3. Create an **OperatorGroup** object YAML file:

   ```
   cat <<EOF | oc apply -f -
   apiVersion: operators.coreos.com/v1
   kind: OperatorGroup
   metadata:
     name: node-observability-operator
     namespace: node-observability-operator
   spec:
     targetNamespaces: []
   EOF
   ```

4. Create a **Subscription** object YAML file to subscribe a namespace to an Operator:

   ```
   cat <<EOF | oc apply -f -
   apiVersion: operators.coreos.com/v1alpha1
   kind: Subscription
   metadata:
     name: node-observability-operator
     namespace: node-observability-operator
   spec:
     channel: alpha
     name: node-observability-operator
     source: redhat-operators
     sourceNamespace: openshift-marketplace
   EOF
   ```

Verification

1. View the install plan name by running the following command:

   ```
   $ oc -n node-observability-operator get sub node-observability-operator -o yaml | yq
   '.status.installplan.name'
   ```

   ### Example output

   ```
   install-dt54w
   ```

2. Verify the install plan status by running the following command:

   ```
   $ oc -n node-observability-operator get ip <install_plan_name> -o yaml | yq '.status.phase'
   ```

   <install_plan_name> is the install plan name that you obtained from the output of the previous command.

Example output

```
COMPLETE
```

3. Verify that the Node Observability Operator is up and running:

```
$ oc get deploy -n node-observability-operator
```

Example output

```
NAME                                        READY   UP-TO-DATE  AVAILABLE   AGE
node-observability-operator-controller-manager  1/1     1           1           40h
```

16.2.2. 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.

16.3. 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 matching the **nodeSelector**.

IMPORTANT

If CRI-O profiling is not enabled on the worker nodes, the **NodeObservabilityMachineConfig** resource gets created. Worker nodes matching the **nodeSelector** specified in **NodeObservability** CR restarts. This might take 10 or more minutes to complete.

NOTE

Kubelet profiling is enabled by default.

The CRI-O unix socket of the node is mounted on the agent pod, which allows the agent to communicate with CRI-O to run the pprof request. Similarly, the **kubelet-serving-ca** certificate chain is mounted on the agent pod, which allows secure communication between the agent and node's kubelet endpoint.

Prerequisites

- You have installed the Node Observability Operator.
- You have installed the OpenShift CLI (oc).
- You have access to the cluster with **cluster-admin** privileges.

Procedure

1. Log in to the OpenShift Container Platform CLI by running the following command:

```
$ oc login -u kubeadmin https://<HOSTNAME>:6443
```

2. Switch back to the **node-observability-operator** namespace by running the following command:

```
$ oc project node-observability-operator
```

3. Create a CR file named **nodeobservability.yaml** that contains the following text:

```
apiVersion: nodeobservability.olm.openshift.io/v1alpha2
kind: NodeObservability
metadata:
  name: cluster (1)
spec:
  nodeSelector:
    kubernetes.io/hostname: <node_hostname> (2)
  type: crio-kubelet
```

(1) You must specify the name as **cluster** because there should be only one **NodeObservability** CR per cluster.

2 Specify the nodes on which the Node Observability agent must be deployed.

4. Run the **NodeObservability** CR:

```
oc apply -f nodeobservability.yaml
```

Example output

```
nodeobservability.olm.openshift.io/cluster created
```

5. Review the status of the **NodeObservability** CR by running the following command:

```
$ oc get nob/cluster -o yaml | 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**.

16.4. 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. The lifetime of data is bound to the agent pod through the **emptyDir** volume, so you can access the profiling data while the agent pod is in the **running** status.

IMPORTANT

You can request only one profiling query at any point of time.

Prerequisites

- You have installed the Node Observability Operator.
- You have created the **NodeObservability** custom resource (CR).
- You have access to the cluster with **cluster-admin** privileges.

Procedure

1. Create a **NodeObservabilityRun** resource file named **nodeobservabilityrun.yaml** that contains the following text:

```
apiVersion: nodeobservability.olm.openshift.io/v1alpha2
kind: NodeObservabilityRun
metadata:
  name: nodeobservabilityrun
spec:
  nodeObservabilityRef:
    name: cluster
```

2. Trigger the profiling query by running the **NodeObservabilityRun** resource:

   ```
   $ oc apply -f nodeobservabilityrun.yaml
   ```

3. Review the status of the **NodeObservabilityRun** by running the following command:

   ```
   $ oc get nodeobservabilityrun nodeobservabilityrun -o yaml  | 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
   ```

CHAPTER 17. CLUSTERS AT THE NETWORK FAR EDGE

17.1. CHALLENGES OF THE NETWORK FAR EDGE

Edge computing presents complex challenges when managing many sites in geographically displaced locations. Use zero touch provisioning (ZTP) and GitOps to provision and manage sites at the far edge of the network.

17.1.1. Overcoming the challenges of the network far edge

Today, service providers want to deploy their infrastructure at the edge of the network. This presents significant challenges:

- How do you handle deployments of many edge sites in parallel?
- What happens when you need to deploy sites in disconnected environments?
- How do you manage the lifecycle of large fleets of clusters?

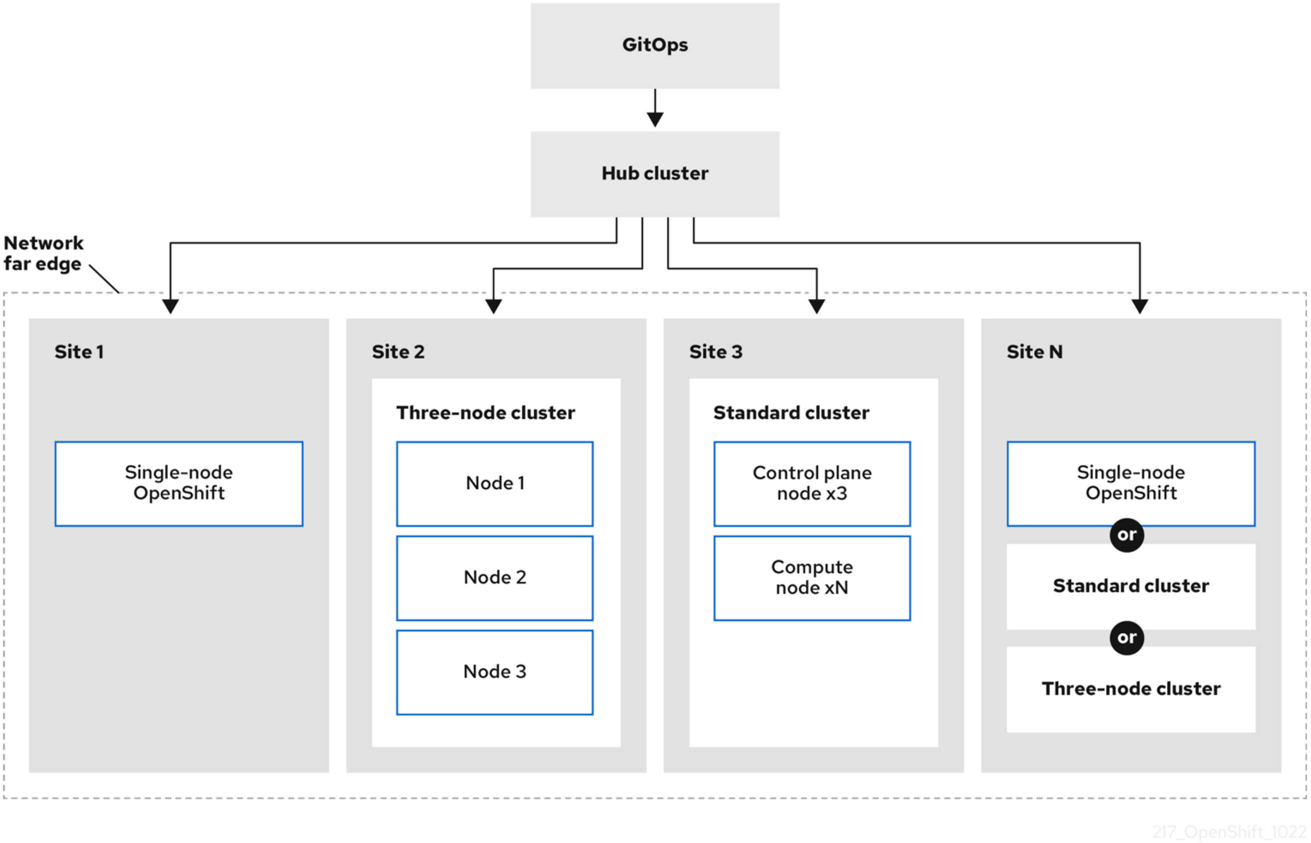
Zero touch provisioning (ZTP) and *GitOps* meets these challenges by allowing you to provision remote edge sites at scale with declarative site definitions and configurations for bare-metal equipment. Template or overlay configurations install OpenShift Container Platform features that are required for CNF workloads. The full lifecycle of installation and upgrades is handled through the ZTP pipeline.

ZTP uses GitOps for infrastructure deployments. With GitOps, you use declarative YAML files and other defined patterns stored in Git repositories. Red Hat Advanced Cluster Management (RHACM) uses your Git repositories to drive the deployment of your infrastructure.

GitOps provides traceability, role-based access control (RBAC), and a single source of truth for the desired state of each site. Scalability issues are addressed by Git methodologies and event driven operations through webhooks.

You start the ZTP workflow by creating declarative site definition and configuration custom resources (CRs) that the ZTP pipeline delivers to the edge nodes.

The following diagram shows how ZTP works within the far edge framework.



17.1.2. Using ZTP to provision clusters at the network far edge

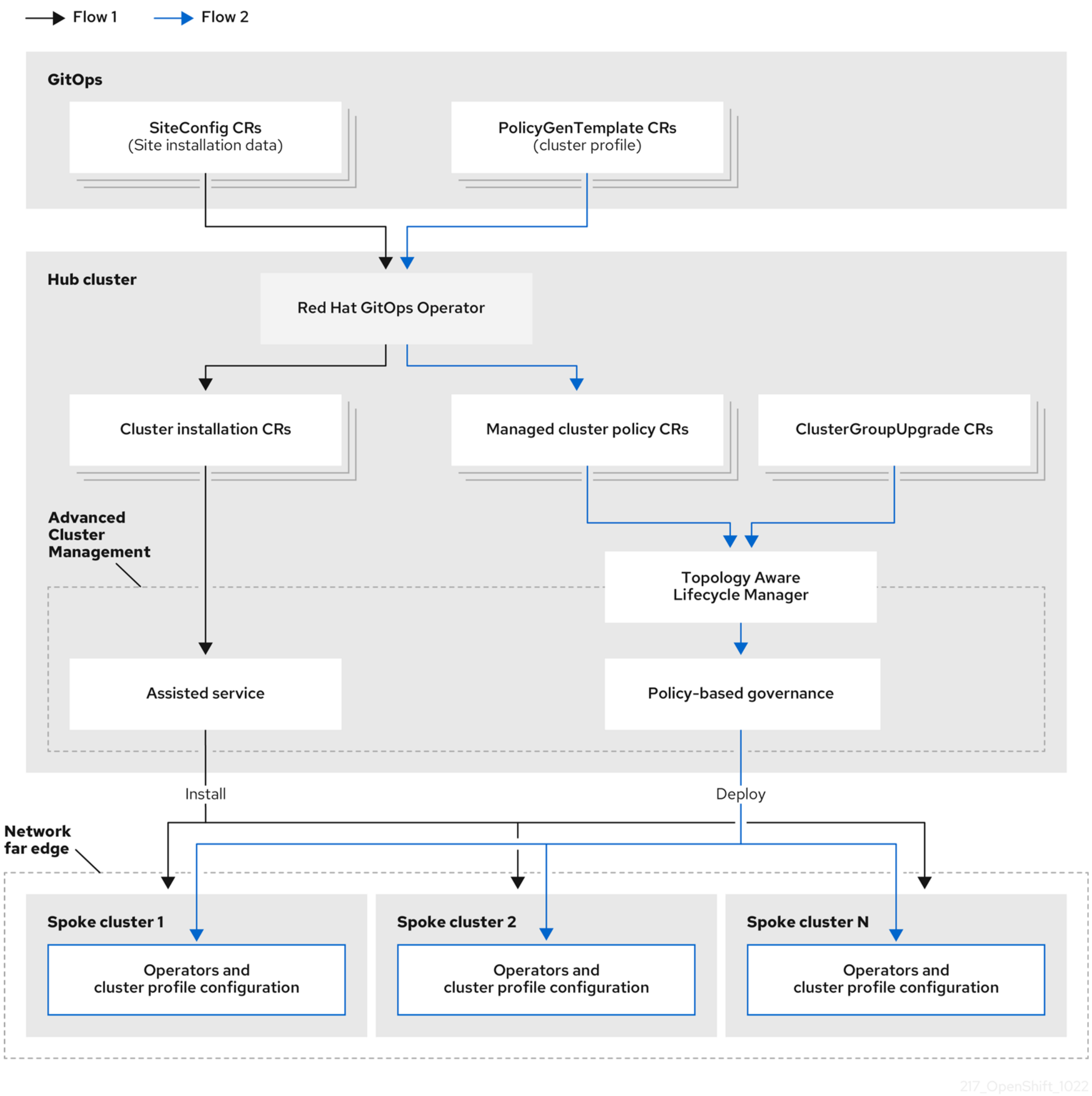
Red Hat Advanced Cluster Management (RHACM) manages clusters in a hub-and-spoke architecture, where a single hub cluster manages many spoke clusters. Hub clusters running RHACM provision and deploy the managed clusters by using zero touch provisioning (ZTP) and the assisted service that is deployed when you install RHACM.

The assisted service handles provisioning of OpenShift Container Platform on single node clusters, three-node clusters, or standard clusters running on bare metal.

A high-level overview of using ZTP to provision and maintain bare-metal hosts with OpenShift Container Platform is as follows:

- A hub cluster running RHACM manages an OpenShift image registry that mirrors the OpenShift Container Platform release images. RHACM uses the OpenShift image registry to provision the managed clusters.
- You manage the bare-metal hosts in a YAML format inventory file, versioned in a Git repository.
- You make the hosts ready for provisioning as managed clusters, and use RHACM and the assisted service to install the bare-metal hosts on site.

Installing and deploying the clusters is a two-stage process, involving an initial installation phase, and a subsequent configuration phase. The following diagram illustrates this workflow:



17.1.3. Installing managed clusters with SiteConfig resources and RHACM

GitOps ZTP uses **SiteConfig** custom resources (CRs) in a Git repository to manage the processes that install OpenShift Container Platform clusters. The **SiteConfig** CR contains cluster-specific parameters required for installation. It has options for applying select configuration CRs during installation including user defined extra manifests.

The ZTP GitOps plugin processes **SiteConfig** CRs to generate a collection of CRs on the hub cluster. This triggers the assisted service in Red Hat Advanced Cluster Management (RHACM) to install OpenShift Container Platform on the bare-metal host. You can find installation status and error messages in these CRs on the hub cluster.

You can provision single clusters manually or in batches with ZTP:

Provisioning a single cluster

Create a single **SiteConfig** CR and related installation and configuration CRs for the cluster, and apply them in the hub cluster to begin cluster provisioning. This is a good way to test your CRs before deploying on a larger scale.

Provisioning many clusters

Install managed clusters in batches of up to 400 by defining **SiteConfig** and related CRs in a Git repository. ArgoCD uses the **SiteConfig** CRs to deploy the sites. The RHACM policy generator creates the manifests and applies them to the hub cluster. This starts the cluster provisioning process.

17.1.4. Configuring managed clusters with policies and PolicyGenTemplate resources

Zero touch provisioning (ZTP) uses Red Hat Advanced Cluster Management (RHACM) to configure clusters by using a policy-based governance approach to applying the configuration.

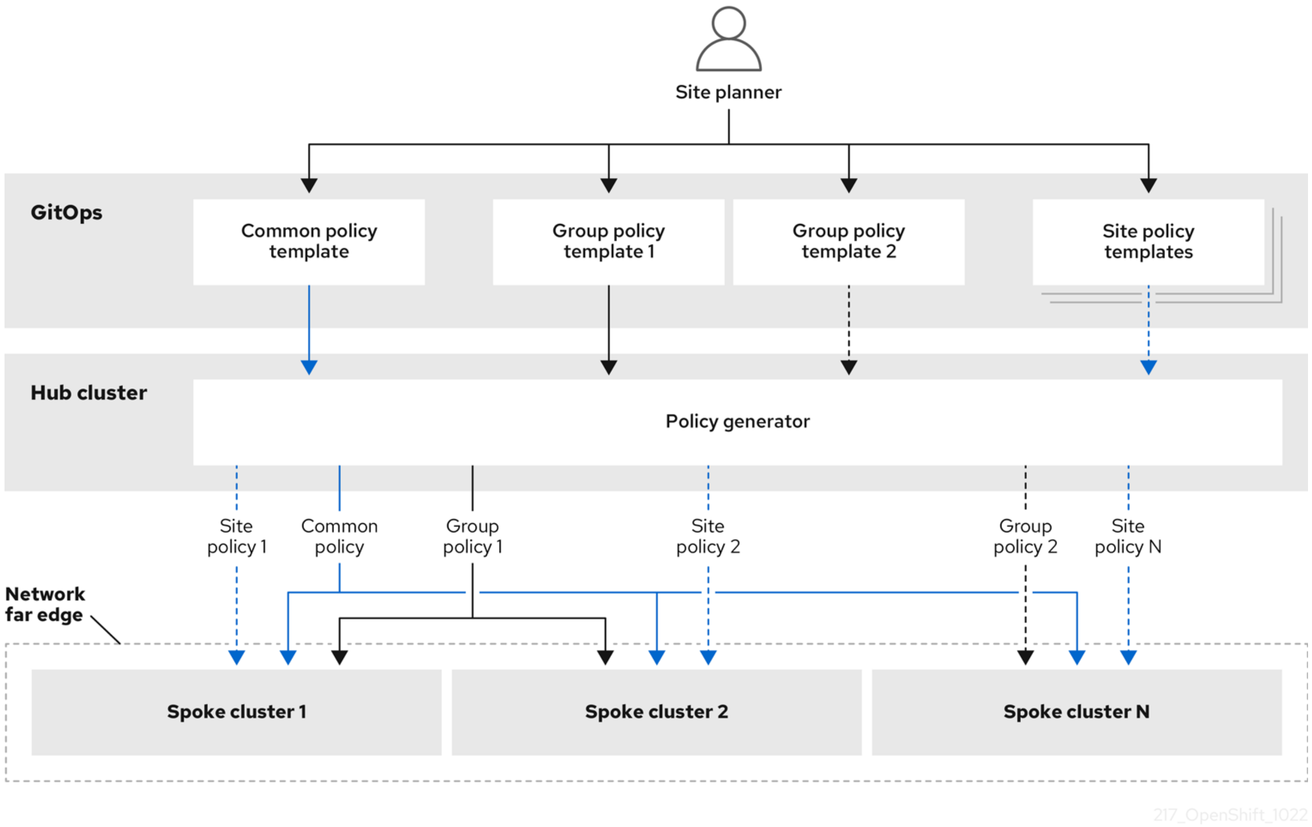
The policy generator or **PolicyGen** is a plugin for the GitOps Operator that enables the creation of RHACM policies from a concise template. The tool can combine multiple CRs into a single policy, and you can generate multiple policies that apply to various subsets of clusters in your fleet.

NOTE

For scalability and to reduce the complexity of managing configurations across the fleet of clusters, use configuration CRs with as much commonality as possible.

- Where possible, apply configuration CRs using a fleet-wide common policy.
- The next preference is to create logical groupings of clusters to manage as much of the remaining configurations as possible under a group policy.
- When a configuration is unique to an individual site, use RHACM templating on the hub cluster to inject the site-specific data into a common or group policy. Alternatively, apply an individual site policy for the site.

The following diagram shows how the policy generator interacts with GitOps and RHACM in the configuration phase of cluster deployment.



For large fleets of clusters, it is typical for there to be a high-level of consistency in the configuration of those clusters.

The following recommended structuring of policies combines configuration CRs to meet several goals:

- Describe common configurations once and apply to the fleet.
- Minimize the number of maintained and managed policies.
- Support flexibility in common configurations for cluster variants.

Table 17.1. Recommended PolicyGenTemplate policy categories

Policy category	Description
Common	A policy that exists in the common category is applied to all clusters in the fleet. Use common **PolicyGenTemplate** CRs to apply common installation settings across all cluster types.
Groups	A policy that exists in the groups category is applied to a group of clusters in the fleet. Use group **PolicyGenTemplate** CRs to manage specific aspects of single-node, three-node, and standard cluster installations. Cluster groups can also follow geographic region, hardware variant, etc.
Sites	A policy that exists in the sites category is applied to a specific cluster site. Any cluster can have its own specific policies maintained.

Additional resources

- For more information about extracting the reference **SiteConfig** and **PolicyGenTemplate** CRs from the **ztp-site-generate** container image, see Preparing the ZTP Git repository.

17.2. PREPARING THE HUB CLUSTER FOR ZTP

To use RHACM in a disconnected environment, create a mirror registry that mirrors the OpenShift Container Platform release images and Operator Lifecycle Manager (OLM) catalog that contains the required Operator images. OLM manages, installs, and upgrades Operators and their dependencies in the cluster. You can also use a disconnected mirror host to serve the RHCOS ISO and RootFS disk images that are used to provision the bare-metal hosts.

17.2.1. Telco RAN 4.12 validated solution software versions

The Red Hat Telco Radio Access Network (RAN) version 4.12 solution has been validated using the following Red Hat software products.

Table 17.2. Telco RAN 4.12 validated solution software

Product	Software version
Hub cluster OpenShift Container Platform version	4.12
GitOps ZTP plugin	4.10, 4.11, or 4.12
Red Hat Advanced Cluster Management (RHACM)	2.6, 2.7
Red Hat OpenShift GitOps	1.9, 1.10
Topology Aware Lifecycle Manager (TALM)	4.10, 4.11, or 4.12

17.2.2. Installing GitOps ZTP in a disconnected environment

Use Red Hat Advanced Cluster Management (RHACM), Red Hat OpenShift GitOps, and Topology Aware Lifecycle Manager (TALM) on the hub cluster in the disconnected environment to manage the deployment of multiple managed clusters.

Prerequisites

- You have installed the OpenShift Container Platform CLI (**oc**).
- You have logged in as a user with **cluster-admin** privileges.
- You have configured a disconnected mirror registry for use in the cluster.

NOTE

The disconnected mirror registry that you create must contain a version of TALM backup and pre-cache images that matches the version of TALM running in the hub cluster. The spoke cluster must be able to resolve these images in the disconnected mirror registry.

Procedure

- Install RHACM in the hub cluster. See Installing RHACM in a disconnected environment .
- Install GitOps and TALM in the hub cluster.

Additional resources

- Installing OpenShift GitOps
- Installing TALM
- Mirroring an Operator catalog

17.2.3. Adding RHCOS ISO and RootFS images to the disconnected mirror host

Before you begin installing clusters in the disconnected environment with Red Hat Advanced Cluster Management (RHACM), you must first host Red Hat Enterprise Linux CoreOS (RHCOS) images for it to use. Use a disconnected mirror to host the RHCOS images.

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 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 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> 1
    ```

    ```
    $ export ROOTFS_IMAGE_NAME=<rootfs_image_name> 1
    ```

    ```
    $ export OCP_VERSION=<ocp_version> 1
    ```

    1 ISO image name, for example, **rhcos-4.12.1-x86_64-live.x86_64.iso**

    1 RootFS image name, for example, **rhcos-4.12.1-x86_64-live-rootfs.x86_64.img**

1 OpenShift Container Platform version, for example, **4.12.1**

b. Download the required images:

```
$ sudo wget https://mirror.openshift.com/pub/openshift-
v4/dependencies/rhcos/4.12/${OCP_VERSION}/${ISO_IMAGE_NAME} -O
/var/www/html/${ISO_IMAGE_NAME}
```

```
$ sudo wget https://mirror.openshift.com/pub/openshift-
v4/dependencies/rhcos/4.12/${OCP_VERSION}/${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}
```

Example output

```
Saving to: rhcos-4.12.1-x86_64-live.x86_64.iso
rhcos-4.12.1-x86_64-live.x86_64.iso-  11%[====>    ]  10.01M  4.71MB/s
```

Additional resources

- Creating a mirror registry
- Mirroring images for a disconnected installation

17.2.4. Enabling the assisted service

Red Hat Advanced Cluster Management (RHACM) uses the assisted service to deploy OpenShift Container Platform clusters. The assisted service is deployed automatically when you enable the MultiClusterHub Operator on Red Hat Advanced Cluster Management (RHACM). After that, you need to configure the **Provisioning** resource to watch all namespaces and to update the **AgentServiceConfig** custom resource (CR) with references to the ISO and RootFS images that are hosted on the mirror registry HTTP server.

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 RHACM with MultiClusterHub enabled.

Procedure

1. Enable the **Provisioning** resource to watch all namespaces and configure mirrors for disconnected environments. For more information, see Enabling the Central Infrastructure Management service.

2. Update the **AgentServiceConfig** CR by running the following command:

```
$ oc edit AgentServiceConfig
```

3. Add the following entry to the **items.spec.osImages** field in the CR:

```
- cpuArchitecture: x86_64
    openshiftVersion: "4.12"
    rootFSUrl: https://<host>/<path>/rhcos-live-rootfs.x86_64.img
    url: https://<mirror-registry>/<path>/rhcos-live.x86_64.iso
```

where:

<host>

Is the fully qualified domain name (FQDN) for the target mirror registry HTTP server.

<path>

Is the path to the image on the target mirror registry.

Save and quit the editor to apply the changes.

17.2.5. Configuring the hub cluster to use a disconnected mirror registry

You can configure the hub cluster to use a disconnected mirror registry for a disconnected environment.

Prerequisites

- You have a disconnected hub cluster installation with Red Hat Advanced Cluster Management (RHACM) 2.7 installed.
- You have hosted the **rootfs** and **iso** images on an HTTP server. See the *Additional resources* section for guidance about *Mirroring the OpenShift Container Platform image repository* .

> **WARNING**
>
> If you enable TLS for the HTTP 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 managed clusters and the HTTP 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

1. Create a **ConfigMap** containing the mirror registry config:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: assisted-installer-mirror-config
  namespace: multicluster-engine 1
  labels:
    app: assisted-service
data:
  ca-bundle.crt: <certificate> 2
  registries.conf: | 3
    unqualified-search-registries = ["registry.access.redhat.com", "docker.io"]

    [[registry]]
      location = <mirror_registry_url> 4
      insecure = false
      mirror-by-digest-only = true
```

1 The **ConfigMap** namespace must be set to **multicluster-engine**.

2 The mirror registry's certificate used when creating the mirror registry.

3 The configuration file for the mirror registry. The mirror registry configuration adds mirror information to **/etc/containers/registries.conf** in the Discovery image. The mirror information is stored in the **imageContentSources** section of the **install-config.yaml** file when passed to the installation program. The Assisted Service pod running on the HUB cluster fetches the container images from the configured mirror registry.

4 The URL of the mirror registry. You must use the URL from the **imageContentSources** section by running the **oc adm release mirror** command when you configure the mirror registry. For more information, see the *Mirroring the OpenShift Container Platform image repository* section.

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: multicluster-engine 1
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 2
```

```
osImages:
  - openshiftVersion: <ocp_version>
    url: <iso_url> 3
```

1 Set the **AgentServiceConfig** namespace to **multicluster-engine** to match the **ConfigMap** namespace

2 Set **mirrorRegistryRef.name** to match the definition specified in the related **ConfigMap** CR

3 Set the URL for the ISO hosted on 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.

Additional resources

- Mirroring the OpenShift Container Platform image repository

17.2.6. Configuring the hub cluster to use unauthenticated registries

You can configure the hub cluster to use unauthenticated registries. Unauthenticated registries does not require authentication to access and download images.

Prerequisites

- You have installed and configured a hub cluster and installed Red Hat Advanced Cluster Management (RHACM) on the hub cluster.
- You have installed the OpenShift Container Platform CLI (oc).
- You have logged in as a user with **cluster-admin** privileges.
- You have configured an unauthenticated registry for use with the hub cluster.

Procedure

1. Update the **AgentServiceConfig** custom resource (CR) by running the following command:

```
$ oc edit AgentServiceConfig agent
```

2. Add the **unauthenticatedRegistries** field in the CR:

```
apiVersion: agent-install.openshift.io/v1beta1
kind: AgentServiceConfig
metadata:
  name: agent
spec:
  unauthenticatedRegistries:
```

```
- example.registry.com
- example.registry2.com
...
```

Unauthenticated registries are listed under **spec.unauthenticatedRegistries** in the **AgentServiceConfig** resource. Any registry on this list is not required to have an entry in the pull secret used for the spoke cluster installation. **assisted-service** validates the pull secret by making sure it contains the authentication information for every image registry used for installation.

NOTE

Mirror registries are automatically added to the ignore list and do not need to be added under **spec.unauthenticatedRegistries**. Specifying the **PUBLIC_CONTAINER_REGISTRIES** environment variable in the **ConfigMap** overrides the default values with the specified value. The **PUBLIC_CONTAINER_REGISTRIES** defaults are quay.io and registry.svc.ci.openshift.org.

Verification

Verify that you can access the newly added registry from the hub cluster by running the following commands:

1. Open a debug shell prompt to the hub cluster:

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

2. Test access to the unauthenticated registry by running the following command:

```
sh-4.4# podman login -u kubeadmin -p $(oc whoami -t) <unauthenticated_registry>
```

where:

<unauthenticated_registry>

Is the new registry, for example, **unauthenticated-image-registry.openshift-image-registry.svc:5000**.

Example output

```
Login Succeeded!
```

17.2.7. Configuring the hub cluster with ArgoCD

You can configure the hub cluster with a set of ArgoCD applications that generate the required installation and policy custom resources (CRs) for each site with GitOps zero touch provisioning (ZTP).

NOTE

Red Hat Advanced Cluster Management (RHACM) uses **SiteConfig** CRs to generate the Day 1 managed cluster installation CRs for ArgoCD. Each ArgoCD application can manage a maximum of 300 **SiteConfig** CRs.

Prerequisites

- You have a OpenShift Container Platform hub cluster with Red Hat Advanced Cluster Management (RHACM) and Red Hat OpenShift GitOps installed.

- You have extracted the reference deployment from the ZTP GitOps plugin container as described in the "Preparing the GitOps ZTP site configuration repository" section. Extracting the reference deployment creates the **out/argocd/deployment** directory referenced in the following procedure.

Procedure

1. 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 GitOps ZTP site configuration 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 ends with **.git**, for example, **https://repo.example.com/repo.git**.

      - The **targetRevision** indicates which Git repository branch to monitor.

      - **path** specifies the path to the **SiteConfig** and **PolicyGenTemplate** CRs, respectively.

2. To install the ZTP GitOps plugin you must patch the ArgoCD instance in the hub cluster by using the patch file previously extracted into the **out/argocd/deployment/** directory. Run the following command:

```
$ oc patch argocd openshift-gitops \
-n openshift-gitops --type=merge \
--patch-file out/argocd/deployment/argocd-openshift-gitops-patch.json
```

> NOTE
>
> For a disconnected environment, amend the **out/argocd/deployment/argocd-openshift-gitops-patch.json** file with the **ztp-site-generate** image mirrored in your local registry. Run the following command:
>
> ```
> $ oc patch argocd openshift-gitops -n openshift-gitops --type='json' \
> -p='[{"op": "replace", "path": "/spec/repo/initContainers/0/image", \
> "value": "<local_registry>/<ztp_site_generate_image_ref>"}]'
> ```
>
> where:
>
> **<local_registry>**
>
> Is the URL of the disconnected registry, for example, **my.local.registry:5000**
>
> **<ztp-site-generate-image-ref>**
>
> Is the path to the mirrored **ztp-site-generate** image in the local registry, for example **openshift4-ztp-site-generate:custom**.

3. In RHACM 2.7 and later, the multicluster engine enables the **cluster-proxy-addon** feature by default. To disable this feature, apply the following patch to disable and remove the relevant hub cluster and managed cluster pods that are responsible for this add-on.

   ```
   $ oc patch multiclusterengines.multicluster.openshift.io multiclusterengine --type=merge --
   patch-file out/argocd/deployment/disable-cluster-proxy-addon.json
   ```

4. Apply the pipeline configuration to your hub cluster by using the following command:

   ```
   $ oc apply -k out/argocd/deployment
   ```

17.2.8. Preparing the GitOps ZTP site configuration repository

Before you can use the ZTP GitOps pipeline, you need to prepare the Git repository to host the site configuration data.

Prerequisites

- You have configured the hub cluster GitOps applications for generating the required installation and policy custom resources (CRs).
- You have deployed the managed clusters using zero touch provisioning (ZTP).

Procedure

1. Create a directory structure with separate paths for the **SiteConfig** and **PolicyGenTemplate** CRs.
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:v4.12
   ```

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

```
$ podman run --log-driver=none --rm registry.redhat.io/openshift4/ztp-site-generate-
rhel8:v4.12 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.

17.3. INSTALLING MANAGED CLUSTERS WITH RHACM AND SITECONFIG RESOURCES

You can provision OpenShift Container Platform clusters at scale with Red Hat Advanced Cluster Management (RHACM) using the assisted service and the GitOps plugin policy generator with core-reduction technology enabled. The zero touch priovisioning (ZTP) pipeline performs the cluster installations. ZTP can be used in a disconnected environment.

17.3.1. 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), the assisted service, and the Topology Aware Lifecycle Manager (TALM) use the CRs to install and configure the managed 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 process, after OpenShift installation, the TALM steps through the created **inform** policies and enforces them on the target managed cluster(s). This applies the configuration to the managed cluster. Outside of the ZTP phase of the cluster lifecycle, this allows you to change policies without the risk of immediately rolling those changes out to affected managed clusters. You can control the timing and the set of remediated clusters by using TALM.

Automatic creation of ClusterGroupUpgrade CRs

To automate the initial configuration of newly deployed clusters, TALM monitors the state of all **ManagedCluster** CRs on the hub cluster. Any **ManagedCluster** CR that 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. The wave annotation is not used other than for 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 postinstallation 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 that 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. The **ztp-done** label affects automatic **ClusterGroupUpgrade** CR creation by TALM. Do not manipulate this label after the initial ZTP installation of the cluster.

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.

17.3.2. Overview of deploying managed clusters with ZTP

Red Hat Advanced Cluster Management (RHACM) uses zero touch provisioning (ZTP) to deploy single-node OpenShift Container Platform clusters, three-node clusters, and standard clusters. You manage site configuration data as OpenShift Container Platform custom resources (CRs) in a Git repository. ZTP uses a declarative GitOps approach for a develop once, deploy anywhere model to deploy the managed 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
- 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

Overview of the managed site installation process
After you apply the managed site custom resources (CRs) on the hub cluster, the following actions happen automatically:

1. A Discovery image ISO file is generated and booted on the target host.
2. When the ISO file successfully boots on the target host it reports the host hardware information to RHACM.
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 is complete when the **Agent** CR for the managed cluster is created on the hub cluster.

IMPORTANT

The target bare-metal host must meet the networking, firmware, and hardware requirements listed in Recommended single-node OpenShift cluster configuration for vDU application workloads.

17.3.3. Creating the managed bare-metal host secrets

Add the required **Secret** custom resources (CRs) for the managed bare-metal host to the hub cluster. You need a secret for the ZTP pipeline to access the Baseboard Management Controller (BMC) and a secret for the assisted installer service to pull cluster installation images from the registry.

NOTE

The secrets are referenced from the **SiteConfig** CR by name. The namespace must match the **SiteConfig** namespace.

Procedure

1. Create a YAML secret file containing credentials for the host Baseboard Management Controller (BMC) and a pull secret required for installing OpenShift and all add-on cluster Operators:
   a. Save the following YAML as the file **example-sno-secret.yaml**:

```
apiVersion: v1
```

```
kind: Secret
metadata:
  name: example-sno-bmc-secret
  namespace: example-sno ①
data: ②
  password: <base64_password>
  username: <base64_username>
type: Opaque
---
apiVersion: v1
kind: Secret
metadata:
  name: pull-secret
  namespace: example-sno ③
data:
  .dockerconfigjson: <pull_secret> ④
type: kubernetes.io/dockerconfigjson
```

① Must match the namespace configured in the related **SiteConfig** CR

② Base64-encoded values for **password** and **username**

③ Must match the namespace configured in the related **SiteConfig** CR

④ Base64-encoded pull secret

2. Add the relative path to **example-sno-secret.yaml** to the **kustomization.yaml** file that you use to install the cluster.

17.3.4. Configuring Discovery ISO kernel arguments for installations using GitOps ZTP

The GitOps ZTP workflow uses the Discovery ISO as part of the OpenShift Container Platform installation process on managed bare-metal hosts. You can edit the **InfraEnv** resource to specify kernel arguments for the Discovery ISO. This is useful for cluster installations with specific environmental requirements. For example, configure the **rd.net.timeout.carrier** kernel argument for the Discovery ISO to facilitate static networking for the cluster or to receive a DHCP address before downloading the root file system during installation.

NOTE

In OpenShift Container Platform 4.12, you can only add kernel arguments. You can not replace or delete kernel arguments.

Prerequisites

- You have installed the OpenShift CLI (oc).
- You have logged in to the hub cluster as a user with cluster-admin privileges.

Procedure

1. Create the **InfraEnv** CR and edit the **spec.kernelArguments** specification to configure kernel arguments.

a. Save the following YAML in an **InfraEnv-example.yaml** file:

NOTE

The **InfraEnv** CR in this example uses template syntax such as **{{ .Cluster.ClusterName }}** that is populated based on values in the **SiteConfig** CR. The **SiteConfig** CR automatically populates values for these templates during deployment. Do not edit the templates manually.

```
apiVersion: agent-install.openshift.io/v1beta1
kind: InfraEnv
metadata:
  annotations:
    argocd.argoproj.io/sync-wave: "1"
  name: "{{ .Cluster.ClusterName }}"
  namespace: "{{ .Cluster.ClusterName }}"
spec:
  clusterRef:
    name: "{{ .Cluster.ClusterName }}"
    namespace: "{{ .Cluster.ClusterName }}"
  kernelArguments:
    - operation: append 1
      value: audit=0 2
    - operation: append
      value: trace=1
  sshAuthorizedKey: "{{ .Site.SshPublicKey }}"
  proxy: "{{ .Cluster.ProxySettings }}"
  pullSecretRef:
    name: "{{ .Site.PullSecretRef.Name }}"
  ignitionConfigOverride: "{{ .Cluster.IgnitionConfigOverride }}"
  nmStateConfigLabelSelector:
    matchLabels:
      nmstate-label: "{{ .Cluster.ClusterName }}"
  additionalNTPSources: "{{ .Cluster.AdditionalNTPSources }}"
```

1 Specify the append operation to add a kernel argument.

2 Specify the kernel argument you want to configure. This example configures the audit kernel argument and the trace kernel argument.

2. Commit the **InfraEnv-example.yaml** CR to the same location in your Git repository that has the **SiteConfig** CR and push your changes. The following example shows a sample Git repository structure:

```
~/example-ztp/install
          └── site-install
               ├── siteconfig-example.yaml
               ├── InfraEnv-example.yaml
               ...
```

3. Edit the **spec.clusters.crTemplates** specification in the **SiteConfig** CR to reference the **InfraEnv-example.yaml** CR in your Git repository:

```
clusters:
  crTemplates:
    InfraEnv: "InfraEnv-example.yaml"
```

When you are ready to deploy your cluster by committing and pushing the **SiteConfig** CR, the build pipeline uses the custom **InfraEnv-example** CR in your Git repository to configure the infrastructure environment, including the custom kernel arguments.

Verification

To verify that the kernel arguments are applied, after the Discovery image verifies that OpenShift Container Platform is ready for installation, you can SSH to the target host before the installation process begins. At that point, you can view the kernel arguments for the Discovery ISO in the **/proc/cmdline** file.

1. Begin an SSH session with the target host:

```
$ ssh -i /path/to/privatekey core@<host_name>
```

2. View the system's kernel arguments by using the following command:

```
$ cat /proc/cmdline
```

17.3.5. Deploying a managed cluster with SiteConfig and ZTP

Use the following procedure to create a **SiteConfig** custom resource (CR) and related files and initiate the zero touch provisioning (ZTP) cluster deployment.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have logged in to the hub cluster as a user with **cluster-admin** privileges.
- 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 you must configure it as a source repository for the ArgoCD application. See "Preparing the GitOps ZTP site configuration repository" for more information.

> NOTE
>
> When you create the source repository, ensure that you patch the ArgoCD application with the **argocd/deployment/argocd-openshift-gitops-patch.json** patch-file that you extract from the **ztp-site-generate** container. See "Configuring the hub cluster with ArgoCD".

- To be ready for provisioning managed clusters, you require the following for each bare-metal host:

  Network connectivity

Your network requires DNS. Managed cluster hosts should be reachable from the hub cluster. Ensure that Layer 3 connectivity exists between the hub cluster and the managed cluster host.

Baseboard Management Controller (BMC) details

ZTP uses BMC username and password details to connect to the BMC during cluster installation. The GitOps ZTP plugin manages the **ManagedCluster** CRs on the hub cluster based on the **SiteConfig** CR in your site Git repo. You create individual **BMCSecret** CRs for each host manually.

Procedure

1. Create the required managed cluster secrets on the hub cluster. These resources must be in a namespace with a name matching the cluster name. For example, in **out/argocd/example/siteconfig/example-sno.yaml**, the cluster name and namespace is **example-sno**.

   a. Export the cluster namespace by running the following command:

   ```
   $ export CLUSTERNS=example-sno
   ```

   b. Create the namespace:

   ```
   $ oc create namespace $CLUSTERNS
   ```

2. Create pull secret and BMC **Secret** CRs for the managed cluster. The pull secret must contain all the credentials necessary for installing OpenShift Container Platform and all required Operators. See "Creating the managed bare-metal host secrets" for more information.

   NOTE

   The secrets are referenced from the **SiteConfig** custom resource (CR) by name. The namespace must match the **SiteConfig** namespace.

3. 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. For example:

   Example single-node OpenShift cluster SiteConfig CR

   ```
   apiVersion: ran.openshift.io/v1
   kind: SiteConfig
   metadata:
     name: "<site_name>"
     namespace: "<site_name>"
   ```

```
spec:
  baseDomain: "example.com"
  pullSecretRef:
    name: "assisted-deployment-pull-secret" 1
  clusterImageSetNameRef: "openshift-4.12" 2
  sshPublicKey: "ssh-rsa AAAA..." 3
  clusters:
  - clusterName: "<site_name>"
    networkType: "OVNKubernetes"
    clusterLabels: 4
      common: true
      group-du-sno: ""
      sites : "<site_name>"
    clusterNetwork:
      - cidr: 1001:1::/48
        hostPrefix: 64
    machineNetwork:
      - cidr: 1111:2222:3333:4444::/64
    serviceNetwork:
      - 1001:2::/112
    additionalNTPSources:
      - 1111:2222:3333:4444::2
    #crTemplates:
    #  KlusterletAddonConfig: "KlusterletAddonConfigOverride.yaml" 5
    nodes:
      - hostName: "example-node.example.com" 6
        role: "master"
        bmcAddress: idrac-virtualmedia://<out_of_band_ip>/<system_id>/ 7
        bmcCredentialsName:
          name: "bmh-secret" 8
        bootMACAddress: "AA:BB:CC:DD:EE:11"
        bootMode: "UEFI" 9
        rootDeviceHints:
          wwn: "0x11111000000asd123"
        cpuset: "0-1,52-53" 10
        nodeNetwork: 11
          interfaces:
            - name: eno1
              macAddress: "AA:BB:CC:DD:EE:11"
          config:
            interfaces:
              - name: eno1
                type: ethernet
                state: up
                ipv4:
                  enabled: false
                ipv6: 12
                  enabled: true
                  address:
                  - ip: 1111:2222:3333:4444::aaaa: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. Create the **assisted-deployment-pull-secret** CR with the same namespace as the **SiteConfig** CR.

2. **clusterImageSetNameRef** defines an image set available on the hub cluster. To see the list of supported versions on your hub cluster, run **oc get clusterimagesets**.

3. Configure the SSH public key used to access the cluster.

4. Cluster labels must correspond to the **bindingRules** field in the **PolicyGenTemplate** CRs that you define. For example, **policygentemplates/common-ranGen.yaml** applies to all clusters with **common: true** set, **policygentemplates/group-du-sno-ranGen.yaml** applies to all clusters with **group-du-sno: ""** set.

5. Optional. The CR specifed under **KlusterletAddonConfig** is used to override the default **KlusterletAddonConfig** that is created for the cluster.

6. For single-node deployments, define a single 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. BMC address that you use to access the host. Applies to all cluster types.

8. Name of the **bmh-secret** CR that you separately create with the host BMC credentials. When creating the **bmh-secret** CR, use the same namespace as the **SiteConfig** CR that provisions the host.

9. Configures the boot mode for the host. The default value is **UEFI**. Use **UEFISecureBoot** to enable secure boot on the host.

10. **cpuset** must match the value set in the cluster **PerformanceProfile** CR **spec.cpu.reserved** field for workload partitioning.

11. Specifies the network settings for the node.

12. Configures the IPv6 address for the host. For single-node OpenShift clusters with static IP addresses, the node-specific API and Ingress IPs should be the same.

NOTE

For more information about BMC addressing, see the "Additional resources" section.

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.

d. 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.

4. 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**.

5. Commit the **SiteConfig** CR and associated **kustomization.yaml** changes in your Git repository and push the changes.
   The ArgoCD pipeline detects the changes and begins the managed cluster deployment.

Additional resources

- Preparing the GitOps ZTP site configuration repository
- Configuring the hub cluster with ArgoCD
- Signalling ZTP cluster deployment completion with validator inform policies
- Creating the managed bare-metal host secrets
- BMC addressing

17.3.5.1. Single-node OpenShift SiteConfig CR installation reference

Table 17.3. SiteConfig CR installation options for single-node OpenShift clusters

SiteConfig CR field	Description
metadata.name	Set **name** to **assisted-deployment-pull-secret** and create the **assisted-deployment-pull-secret** CR in the same namespace as the **SiteConfig** CR.
spec.clusterImageSetNameRef	Configure the image set available on the hub cluster for all the clusters in the site. To see the list of supported versions on your hub cluster, run **oc get clusterimagesets**.
installConfigOverrides	Set the **installConfigOverrides** field to enable or disable optional components prior to cluster installation. IMPORTANT Use the reference configuration as specified in the example **SiteConfig** CR. Adding additional components back into the system might require additional reserved CPU capacity.
spec.clusters.clusterLabels	Configure cluster labels to correspond to the **bindingRules** field in the **PolicyGenTemplate** CRs that you define. For example, **policygentemplates/common-ranGen.yaml** applies to all clusters with **common: true** set, **policygentemplates/group-du-sno-ranGen.yaml** applies to all clusters with **group-du-sno: ""** set.

SiteConfig CR field	Description
spec.clusters.crTemplates.KlusterletAddonConfig	Optional. Set **KlusterletAddonConfig** to **KlusterletAddonConfigOverride.yaml to override the default `KlusterletAddonConfig** that is created for the cluster.
spec.clusters.nodes.hostName	For single-node deployments, define a single 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**.
spec.clusters.nodes.bmcAddress	BMC address that you use to access the host. Applies to all cluster types. {ztp} supports iPXE and virtual media booting by using Redfish or IPMI protocols. To use iPXE booting, you must use RHACM 2.8 or later. For more information about BMC addressing, see the "Additional resources" section.
spec.clusters.nodes.bmcAddress	BMC address that you use to access the host. Applies to all cluster types. {ztp} supports iPXE and virtual media booting by using Redfish or IPMI protocols. To use iPXE booting, you must use RHACM 2.8 or later. For more information about BMC addressing, see the "Additional resources" section. NOTE In far edge Telco use cases, only virtual media is supported for use with {ztp}.
spec.clusters.nodes.bmcCredentialsName	Configure the **bmh-secret** CR that you separately create with the host BMC credentials. When creating the **bmh-secret** CR, use the same namespace as the **SiteConfig** CR that provisions the host.
spec.clusters.nodes.bootMode	Set the boot mode for the host to **UEFI**. The default value is **UEFI**. Use **UEFISecureBoot** to enable secure boot on the host.
spec.clusters.nodes.rootDeviceHints	Specifies the device for deployment. Identifiers that are stable across reboots are recommended, for example, **wwn: <disk_wwn>** or **deviceName: /dev/disk/by-path/<device_path>**. For a detailed list of stable identifiers, see the "About root device hints section".
spec.clusters.nodes.diskPartition	Optional. The provided example **diskPartition** is used to configure additional disk partitions.
spec.clusters.nodes.cpuset	Configure **cpuset** to match value that you set in the cluster **PerformanceProfile** CR **spec.cpu.reserved** field for workload partitioning.
spec.clusters.nodes.nodeNetwork	Configure the network settings for the node.
spec.clusters.nodes.nodeNetwork.config.interfaces.ipv6	Configure the IPv6 address for the host. For single-node OpenShift clusters with static IP addresses, the node-specific API and Ingress IPs should be the same.

17.3.6. Monitoring managed cluster installation progress

The ArgoCD pipeline uses the **SiteConfig** CR to generate the cluster configuration CRs and syncs it with the hub cluster. You can monitor the progress of the synchronization in the ArgoCD dashboard.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have logged in to the hub cluster as a user with **cluster-admin** privileges.

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 by running the following commands:

    a. Export the cluster name:

    ```
    $ export CLUSTER=<clusterName>
    ```

    b. Query the **AgentClusterInstall** CR for the managed cluster:

    ```
    $ oc get agentclusterinstall -n $CLUSTER $CLUSTER -o jsonpath='{.status.conditions[?(@.type=="Completed")]}' | jq
    ```

    c. Get the installation events for the cluster:

    ```
    $ curl -sk $(oc get agentclusterinstall -n $CLUSTER $CLUSTER -o jsonpath='{.status.debugInfo.eventsURL}')  | jq '.[-2,-1]'
    ```

17.3.7. Troubleshooting GitOps ZTP by validating the installation CRs

The ArgoCD pipeline uses the **SiteConfig** and **PolicyGenTemplate** custom resources (CRs) 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.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have logged in to the hub cluster as a user with **cluster-admin** privileges.

Procedure

1. 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 steps to troubleshoot the ArgoCD pipeline flow from **SiteConfig** files to the installation CRs.

2. Verify that the **ManagedCluster** CR was generated using the **SiteConfig** CR on the hub cluster:

   ```
   $ oc get managedcluster
   ```

3. If the **ManagedCluster** is missing, check if the **clusters** application failed to synchronize the files from the Git repository to the hub cluster:

   ```
   $ oc describe -n openshift-gitops application clusters
   ```

   a. Check for the **Status.Conditions** field to view the error logs for the managed cluster. For example, setting an invalid value for **extraManifestPath:** in the **SiteConfig** CR raises the following error:

      ```
      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
      ```

   b. Check the **Status.Sync** field. If there are log errors, the **Status.Sync** field 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
      ```

17.3.8. Troubleshooting {ztp} virtual media booting on Supermicro servers

SuperMicro X11 servers do not support virtual media installations when the image is served using the **https** protocol. As a result, single-node OpenShift deployments for this environment fail to boot on the target node. To avoid this issue, log in to the hub cluster and disable Transport Layer Security (TLS) in the **Provisioning** resource. This ensures the image is not served with TLS even though the image address uses the **https** scheme.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have logged in to the hub cluster as a user with **cluster-admin** privileges.

Procedure

1. Disable TLS in the **Provisioning** resource by running the following command:

```
$ oc patch provisioning provisioning-configuration --type merge -p '{"spec":
{"disableVirtualMediaTLS": true}}'
```

2. Continue the steps to deploy your single-node OpenShift cluster.

17.3.9. Removing a managed cluster site from the ZTP pipeline

You can remove a managed site and the associated installation and configuration policy CRs from the ZTP pipeline.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have logged in to the hub cluster as a user with **cluster-admin** privileges.

Procedure

1. Remove a site and the associated CRs by removing the associated **SiteConfig** and **PolicyGenTemplate** files from the **kustomization.yaml** file.
   When you run the ZTP pipeline again, the generated CRs are removed.
2. Optional: If you want to permanently remove a site, you should also remove the **SiteConfig** and site-specific **PolicyGenTemplate** files from the Git repository.
3. Optional: 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.

Additional resources

- For information about removing a cluster, see Removing a cluster from management.

17.3.10. Removing obsolete content from the ZTP pipeline

If a change to the **PolicyGenTemplate** configuration results in obsolete policies, for example, if you rename policies, use the following procedure to remove the obsolete policies.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have logged in to the hub cluster as a user with **cluster-admin** privileges.

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

    Removing zero touch provisioning (ZTP) policies from the Git repository, and as a result also removing them from the hub cluster, does not affect the configuration of the managed cluster. The policy and CRs managed by that policy remains in place on the managed cluster.

4. Optional: As an alternative, after making changes to **PolicyGenTemplate** CRs 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 running the following command:

    ```
    $ oc delete policy -n <namespace> <policy_name>
    ```

17.3.11. Tearing down the ZTP pipeline

You can remove the ArgoCD pipeline and all generated ZTP artifacts.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have logged in to the hub cluster as a user with **cluster-admin** privileges.

Procedure

1. Detach all clusters from Red Hat Advanced Cluster Management (RHACM) on the hub cluster.
2. Delete the **kustomization.yaml** file in the **deployment** directory using the following command:

    ```
    $ oc delete -k out/argocd/deployment
    ```

3. Commit and push your changes to the site repository.

17.4. CONFIGURING MANAGED CLUSTERS WITH POLICIES AND POLICYGENTEMPLATE RESOURCES

Applied policy custom resources (CRs) configure the managed clusters that you provision. You can customize how Red Hat Advanced Cluster Management (RHACM) uses **PolicyGenTemplate** CRs to generate the applied policy CRs.

17.4.1. About the PolicyGenTemplate CRD

The **PolicyGenTemplate** custom resource definition (CRD) tells the **PolicyGen** policy generator what custom resources (CRs) to include in the cluster configuration, how to combine 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** CR (**common-du-ranGen.yaml**) extracted from the **ztp-site-generate** reference container. The **common-du-ranGen.yaml** file defines two Red Hat Advanced Cluster Management (RHACM) policies. The polices manage a collection of configuration

CRs, one for each unique value of **policyName** in the CR. **common-du-ranGen.yaml** creates a single placement binding and a placement rule to bind the policies to clusters based on the labels listed in the **bindingRules** section.

Example PolicyGenTemplate CR - common-du-ranGen.yaml

```
---
apiVersion: ran.openshift.io/v1
kind: PolicyGenTemplate
metadata:
  name: "common"
  namespace: "ztp-common"
spec:
  bindingRules:
    common: "true" 1
  sourceFiles: 2
    - fileName: SriovSubscription.yaml
      policyName: "subscriptions-policy"
    - fileName: SriovSubscriptionNS.yaml
      policyName: "subscriptions-policy"
    - fileName: SriovSubscriptionOperGroup.yaml
      policyName: "subscriptions-policy"
    - fileName: SriovOperatorStatus.yaml
      policyName: "subscriptions-policy"
    - fileName: PtpSubscription.yaml
      policyName: "subscriptions-policy"
    - fileName: PtpSubscriptionNS.yaml
      policyName: "subscriptions-policy"
    - fileName: PtpSubscriptionOperGroup.yaml
      policyName: "subscriptions-policy"
    - fileName: PtpOperatorStatus.yaml
      policyName: "subscriptions-policy"
    - fileName: ClusterLogNS.yaml
      policyName: "subscriptions-policy"
    - fileName: ClusterLogOperGroup.yaml
      policyName: "subscriptions-policy"
    - fileName: ClusterLogSubscription.yaml
      policyName: "subscriptions-policy"
    - fileName: ClusterLogOperatorStatus.yaml
      policyName: "subscriptions-policy"
    - fileName: StorageNS.yaml
      policyName: "subscriptions-policy"
    - fileName: StorageOperGroup.yaml
      policyName: "subscriptions-policy"
    - fileName: StorageSubscription.yaml
      policyName: "subscriptions-policy"
    - fileName: StorageOperatorStatus.yaml
      policyName: "subscriptions-policy"
    - fileName: ReduceMonitoringFootprint.yaml
      policyName: "config-policy"
    - fileName: OperatorHub.yaml 3
      policyName: "config-policy"
    - fileName: DefaultCatsrc.yaml 4
      policyName: "config-policy" 5
      metadata:
        name: redhat-operators
      spec:
        displayName: disconnected-redhat-operators
        image: registry.example.com:5000/disconnected-redhat-operators/disconnected-redhat-operator-index:v4.9
    - fileName: DisconnectedICSP.yaml
      policyName: "config-policy"
      spec:
        repositoryDigestMirrors:
        - mirrors:
          - registry.example.com:5000
          source: registry.redhat.io
```

1. **common: "true"** applies the policies to all clusters with this label.
2. Files listed under **sourceFiles** create the Operator policies for installed clusters.
3. **OperatorHub.yaml** configures the OperatorHub for the disconnected registry.
4. **DefaultCatsrc.yaml** configures the catalog source for the disconnected registry.
5. **policyName: "config-policy"** configures Operator subscriptions. The **OperatorHub** CR disables the default and this CR replaces **redhat-operators** with a **CatalogSource** CR that points to the disconnected registry.

A **PolicyGenTemplate** CR can be constructed with any number of included CRs. Apply the following example CR in the hub cluster to generate a policy containing a single CR:

```
apiVersion: ran.openshift.io/v1
kind: PolicyGenTemplate
metadata:
  name: "group-du-sno"
  namespace: "ztp-group"
spec:
  bindingRules:
    group-du-sno: ""
  mcp: "master"
  sourceFiles:
    - fileName: PtpConfigSlave.yaml
      policyName: "config-policy"
      metadata:
        name: "du-ptp-slave"
      spec:
        profile:
        - name: "slave"
          interface: "ens5f0"
          ptp4lOpts: "-2 -s --summary_interval -4"
          phc2sysOpts: "-a -r -n 24"
```

Using the source file **PtpConfigSlave.yaml** as an example, the file defines a **PtpConfig** CR. The generated policy for the **PtpConfigSlave** example is named **group-du-sno-config-policy**. The **PtpConfig** CR defined in the generated **group-du-sno-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-sno-config-policy** CR:

```
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: inform
  disabled: false
  policy-templates:
    - objectDefinition:
      apiVersion: policy.open-cluster-management.io/v1
      kind: ConfigurationPolicy
      metadata:
        name: group-du-ptp-config-policy-config
      spec:
        remediationAction: inform
        severity: low
        namespaceselector:
          exclude:
            - kube-*
          include:
            - '*'
        object-templates:
          - complianceType: musthave
           objectDefinition:
            apiVersion: ptp.openshift.io/v1
            kind: PtpConfig
            metadata:
              name: du-ptp-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
                    .....
```

17.4.2. Recommendations when customizing PolicyGenTemplate CRs

Consider the following best practices when customizing site configuration **PolicyGenTemplate** custom resources (CRs):

- Use as few policies as are necessary. Using fewer policies requires less resources. Each additional policy creates overhead for the hub cluster and the deployed managed 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.

- In disconnected environments, use a single catalog source for all Operators by configuring the registry as a single index containing all Operators. Each additional **CatalogSource** CR on the managed 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 recommendations about scaling clusters with RHACM, see Performance and scalability.

NOTE

When managing large numbers of spoke clusters on the hub cluster, minimize the number of policies to reduce resource consumption.

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, and site hierarchy of policies for managing site configuration, it is especially important to combine site-specific configuration into a single policy.

17.4.3. PolicyGenTemplate CRs for RAN deployments

Use **PolicyGenTemplate** (PGT) custom resources (CRs) to customize the configuration applied to the cluster by using the GitOps zero touch provisioning (ZTP) pipeline. The PGT CR allows you to generate one or more policies to manage the set of configuration CRs on your fleet of clusters. The PGT identifies the set of managed CRs, bundles them into policies, builds the policy wrapping around those CRs, and associates the policies with clusters by using label binding rules.

The reference 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 (Radio Access Network) Distributed Unit (DU) applications. Changes or omissions from the baseline configuration can affect feature availability, performance, and resource utilization. Use the reference **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** container. See "Preparing the GitOps ZTP site configuration 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 17.4. PolicyGenTemplate CRs for RAN deployments

PolicyGenTemplate CR	Description
example-multinode-site.yaml	Contains a set of CRs that get applied to multi-node clusters. These CRs configure SR-IOV features typical for RAN installations.
example-sno-site.yaml	Contains a set of CRs that get applied to single-node OpenShift clusters. These CRs configure SR-IOV features typical for RAN installations.
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.
group-du-3node-validator-ranGen.yaml	**PolicyGenTemplate** CR used to generate the various policies required for three-node clusters.
group-du-standard-validator-ranGen.yaml	**PolicyGenTemplate** CR used to generate the various policies required for standard clusters.
group-du-sno-validator-ranGen.yaml	**PolicyGenTemplate** CR used to generate the various policies required for single-node OpenShift clusters.

Additional resources

- Preparing the GitOps ZTP site configuration repository

17.4.4. Customizing a managed cluster with PolicyGenTemplate CRs

Use the following procedure to customize the policies that get applied to the managed cluster that you provision using the zero touch provisioning (ZTP) pipeline.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have logged in to the hub cluster as a user with **cluster-admin** privileges.
- 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. Create a **PolicyGenTemplate** CR for site-specific configuration CRs.
   a. Choose the appropriate example for your CR from the **out/argocd/example/policygentemplates** folder, for example, **example-sno-site.yaml** or **example-multinode-site.yaml**.
   b. Change the **bindingRules** field in the example file to match the site-specific label included in the **SiteConfig** CR. In the example **SiteConfig** file, the site-specific label is **sites: example-sno**.

   NOTE

   Ensure that the labels defined in your **PolicyGenTemplate bindingRules** field correspond to the labels that are defined in the related managed clusters **SiteConfig** CR.

   c. Change the content in the example file to match the desired configuration.
2. Optional: Create a **PolicyGenTemplate** CR for any common configuration CRs that apply to the entire fleet of clusters.
   a. Select the appropriate example for your CR from the **out/argocd/example/policygentemplates** folder, for example, **common-ranGen.yaml**.
   b. Change the content in the example file to match the desired configuration.
3. Optional: Create a **PolicyGenTemplate** CR for any group configuration CRs that apply to the certain groups of clusters in the fleet.
   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.

a. Select the appropriate example for your CR from the **out/argocd/example/policygentemplates** folder, for example, **group-du-sno-ranGen.yaml**.

b. Change the content in the example file to match the desired configuration.

4. Optional. Create a validator inform policy **PolicyGenTemplate** CR to signal when the ZTP installation and configuration of the deployed cluster is complete. For more information, see "Creating a validator inform policy".

5. Define all the policy namespaces in a YAML file similar to the example **out/argocd/example/policygentemplates/ns.yaml** file.

   IMPORTANT

   Do not include the **Namespace** CR in the same file with the **PolicyGenTemplate** CR.

6. Add the **PolicyGenTemplate** CRs and **Namespace** CR to the **kustomization.yaml** file in the generators section, similar to the example shown in **out/argocd/example/policygentemplates/kustomization.yaml**.

7. Commit the **PolicyGenTemplate** CRs, **Namespace** CR, and associated **kustomization.yaml** file in your Git repository and push the changes.
   The ArgoCD pipeline detects the changes and begins the managed cluster deployment. You can push the changes to the **SiteConfig** CR and the **PolicyGenTemplate** CR simultaneously.

Additional resources

- Signalling ZTP cluster deployment completion with validator inform policies

17.4.5. Monitoring managed cluster policy deployment progress

The ArgoCD pipeline uses **PolicyGenTemplate** CRs in Git to generate the RHACM policies and then sync them to the hub cluster. You can monitor the progress of the managed cluster policy synchronization after the assisted service installs OpenShift Container Platform on the managed cluster.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have logged in to the hub cluster as a user with **cluster-admin** privileges.

Procedure

1. 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 by using the following commands:

   ```
   $ export CLUSTER=<clusterName>
   ```

   ```
   $ oc get clustergroupupgrades -n ztp-install $CLUSTER -o jsonpath='{.status.conditions[-1:]}'
   | jq
   ```

   Example output

   ```
   {
     "lastTransitionTime": "2022-11-09T07:28:09Z",
     "message": "Remediating non-compliant policies",
     "reason": "InProgress",
     "status": "True",
     "type": "Progressing"
   }
   ```

2. You can monitor the detailed cluster policy compliance status by using the RHACM dashboard or the command line.

   a. To check policy compliance by using **oc**, run the following command:

      ```
      $ oc get policies -n $CLUSTER
      ```

      Example output

      ```
      NAME                                                     REMEDIATION ACTION   COMPLIANCE STATE
      AGE
      ztp-common.common-config-policy                          inform               Compliant
      3h42m
      ztp-common.common-subscriptions-policy                   inform               NonCompliant
      3h42m
      ztp-group.group-du-sno-config-policy                     inform               NonCompliant
      3h42m
      ztp-group.group-du-sno-validator-du-policy               inform               NonCompliant
      3h42m
      ztp-install.example1-common-config-policy-pjz9s          enforce              Compliant
      167m
      ztp-install.example1-common-subscriptions-policy-zzd9k   enforce              NonCompliant
      164m
      ztp-site.example1-config-policy                          inform               NonCompliant     3h42m
      ztp-site.example1-perf-policy                            inform               NonCompliant     3h42m
      ```

   b. To check policy status from the RHACM web console, perform the following actions:

      i. Click **Governance → Find policies**.

ii. Click on a cluster policy to check it’s status.

When all of the cluster policies become compliant, ZTP installation and configuration for the cluster is complete. The **ztp-done** label is added to the cluster.

In the reference configuration, the final policy that becomes compliant is the one defined in the ***-du-validator-policy** policy. This policy, when compliant on a cluster, ensures that all cluster configuration, Operator installation, and Operator configuration is complete.

17.4.6. 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 managed cluster. This causes the Operator configuration policies to fail to apply because the CRD is not yet applied to the spoke.

17.4.7. Restarting policy reconciliation

You can restart policy reconciliation when unexpected compliance issues occur, for example, when the **ClusterGroupUpgrade** custom resource (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 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 complaint 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 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 update and a new **ClusterGroupUpgrade** CR created for remediation of the policies.

Additional resources

- For information about using Topology Aware Lifecycle Manager (TALM) to construct your own **ClusterGroupUpgrade** CR, see About the ClusterGroupUpgrade CR.

17.4.8. Changing applied managed cluster CRs using policies

You can remove content from a custom resource (CR) that is deployed in a managed cluster through a policy.

By default, all **Policy** CRs created from a **PolicyGenTemplate** CR have the **complianceType** field set to **musthave**. A **musthave** policy without the removed content is still compliant because the CR on the managed cluster has all the specified content. With this configuration, when you remove content from a CR, TALM removes the content from the policy but the content is not removed from the CR on the managed cluster.

With the **complianceType** field to **mustonlyhave**, the policy ensures that the CR on the cluster is an exact match of what is specified in the policy.

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 deployed a managed cluster from a hub cluster running RHACM.
- You have installed Topology Aware Lifecycle Manager on the hub cluster.

Procedure

1. Remove the content that you no longer need from the affected CRs. In this example, the **disableDrain: false** line was removed from the **SriovOperatorConfig** CR.

   #### Example CR

   ```
   apiVersion: sriovnetwork.openshift.io/v1
   kind: SriovOperatorConfig
   metadata:
     name: default
     namespace: openshift-sriov-network-operator
   spec:
     configDaemonNodeSelector:
       "node-role.kubernetes.io/$mcp": ""
     disableDrain: true
     enableInjector: true
     enableOperatorWebhook: true
   ```

2. Change the **complianceType** of the affected policies to **mustonlyhave** in the **group-du-sno-ranGen.yaml** file.

   #### Example YAML

   ```
   # ...
   - fileName: SriovOperatorConfig.yaml
     policyName: "config-policy"
     complianceType: mustonlyhave
   # ...
   ```

3. Create a **ClusterGroupUpdates** CR and specify the clusters that must receive the CR changes::

   #### Example ClusterGroupUpdates CR

   ```
   apiVersion: ran.openshift.io/v1alpha1
   kind: ClusterGroupUpgrade
   metadata:
     name: cgu-remove
     namespace: default
   spec:
     managedPolicies:
       - ztp-group.group-du-sno-config-policy
     enable: false
     clusters:
     - spoke1
   ```

```
   - spoke2
  remediationStrategy:
    maxConcurrency: 2
    timeout: 240
  batchTimeoutAction:
```

4. Create the **ClusterGroupUpgrade** CR by running the following command:

```
$ oc create -f cgu-remove.yaml
```

5. When you are ready to apply the changes, for example, during an appropriate maintenance window, change the value of the **spec.enable** field to **true** by running the following command:

```
$ oc --namespace=default patch clustergroupupgrade.ran.openshift.io/cgu-remove \
--patch '{"spec":{"enable":true}}' --type=merge
```

Verification

1. Check the status of the policies by running the following command:

```
$ oc get <kind> <changed_cr_name>
```

Example output

```
NAMESPACE   NAME                                          REMEDIATION ACTION
COMPLIANCE STATE   AGE
default     cgu-ztp-group.group-du-sno-config-policy           enforce                           17m
default     ztp-group.group-du-sno-config-policy             inform           NonCompliant
15h
```

When the **COMPLIANCE STATE** of the policy is **Compliant**, it means that the CR is updated and the unwanted content is removed.

2. Check that the policies are removed from the targeted clusters by running the following command on the managed clusters:

```
$ oc get <kind> <changed_cr_name>
```

If there are no results, the CR is removed from the managed cluster.

17.4.9. 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 **ManagedClusterJoined** and **ManagedClusterAvailable** conditions in the **ManagedCluster** CR . If the **ManagedCluster** CR does not have these conditions, 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 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) 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 managed cluster is complete.

The validator inform policy ensures the configuration of the 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.

17.5. MANUALLY INSTALLING A SINGLE-NODE OPENSHIFT CLUSTER WITH ZTP

You can deploy a managed single-node OpenShift cluster by using Red Hat Advanced Cluster Management (RHACM) and the assisted service.

NOTE

If you are creating multiple managed clusters, use the **SiteConfig** method described in Deploying far edge sites with ZTP .

IMPORTANT

The target bare-metal host must meet the networking, firmware, and hardware requirements listed in Recommended cluster configuration for vDU application workloads.

17.5.1. Generating ZTP installation and configuration CRs manually

Use the **generator** entrypoint for the **ztp-site-generate** container to generate the site installation and configuration custom resource (CRs) for a cluster based on **SiteConfig** and **PolicyGenTemplate** CRs.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have logged in to the hub cluster as a user with **cluster-admin** privileges.

Procedure

1. Create an output folder by running the following command:

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

2. Export the **argocd** directory from the **ztp-site-generate** container image:

   ```
   $ podman run --log-driver=none --rm registry.redhat.io/openshift4/ztp-site-generate-
   rhel8:v4.12 extract /home/ztp --tar | tar x -C ./out
   ```

   The **./out** directory has the reference **PolicyGenTemplate** and **SiteConfig** CRs in the **out/argocd/example/** folder.

   ### Example output

   ```
   out
    └── argocd
         └── 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
   ```

3. Create an output folder for the site installation CRs:

   ```
   $ mkdir -p ./site-install
   ```

4. Modify the example **SiteConfig** CR for the cluster type that you want to install. Copy **example-sno.yaml** to **site-1-sno.yaml** and modify the CR to match the details of the site and bare-metal host that you want to install, for example:

   ### Example single-node OpenShift cluster SiteConfig CR

   ```
   apiVersion: ran.openshift.io/v1
   kind: SiteConfig
   metadata:
     name: "<site_name>"
     namespace: "<site_name>"
   spec:
     baseDomain: "example.com"
     pullSecretRef:
       name: "assisted-deployment-pull-secret" 1
     clusterImageSetNameRef: "openshift-4.12" 2
     sshPublicKey: "ssh-rsa AAAA..." 3
     clusters:
     - clusterName: "<site_name>"
       networkType: "OVNKubernetes"
       clusterLabels: 4
   ```

```
  common: true
  group-du-sno: ""
  sites : "<site_name>"
clusterNetwork:
  - cidr: 1001:1::/48
    hostPrefix: 64
machineNetwork:
  - cidr: 1111:2222:3333:4444::/64
serviceNetwork:
  - 1001:2::/112
additionalNTPSources:
  - 1111:2222:3333:4444::2
#crTemplates:
#  KlusterletAddonConfig: "KlusterletAddonConfigOverride.yaml" 5
nodes:
  - hostName: "example-node.example.com" 6
    role: "master"
    bmcAddress: idrac-virtualmedia://<out_of_band_ip>/<system_id>/ 7
    bmcCredentialsName:
      name: "bmh-secret" 8
    bootMACAddress: "AA:BB:CC:DD:EE:11"
    bootMode: "UEFI" 9
    rootDeviceHints:
      wwn: "0x11111000000asd123"
    cpuset: "0-1,52-53" 10
    nodeNetwork: 11
      interfaces:
        - name: eno1
          macAddress: "AA:BB:CC:DD:EE:11"
      config:
        interfaces:
          - name: eno1
            type: ethernet
            state: up
            ipv4:
              enabled: false
            ipv6: 12
              enabled: true
              address:
              - ip: 1111:2222:3333:4444::aaaa: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. Create the **assisted-deployment-pull-secret** CR with the same namespace as the **SiteConfig** CR.

2. **clusterImageSetNameRef** defines an image set available on the hub cluster. To see the list of supported versions on your hub cluster, run **oc get clusterimagesets**.

3. Configure the SSH public key used to access the cluster.

4. Cluster labels must correspond to the **bindingRules** field in the **PolicyGenTemplate** CRs that you define. For example, **policygentemplates/common-ranGen.yaml** applies to all clusters with **common: true** set, **policygentemplates/group-du-sno-ranGen.yaml** applies to all clusters with **group-du-sno: ""** set.

5. Optional. The CR specifed under **KlusterletAddonConfig** is used to override the default **KlusterletAddonConfig** that is created for the cluster.

6. For single-node deployments, define a single 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. BMC address that you use to access the host. Applies to all cluster types.

8. Name of the **bmh-secret** CR that you separately create with the host BMC credentials. When creating the **bmh-secret** CR, use the same namespace as the **SiteConfig** CR that provisions the host.

9. Configures the boot mode for the host. The default value is **UEFI**. Use **UEFISecureBoot** to enable secure boot on the host.

10. **cpuset** must match the value set in the cluster **PerformanceProfile** CR **spec.cpu.reserved** field for workload partitioning.

11. Specifies the network settings for the node.

12. Configures the IPv6 address for the host. For single-node OpenShift clusters with static IP addresses, the node-specific API and Ingress IPs should be the same.

5. Generate the day-0 installation CRs by processing the modified **SiteConfig** CR **site-1-sno.yaml** by running the following command:

```
$ podman run -it --rm -v `pwd`/out/argocd/example/siteconfig:/resources:Z -v `pwd`/site-install:/output:Z,U registry.redhat.io/openshift4/ztp-site-generate-rhel8:v4.12.1 generator install site-1-sno.yaml /output
```

Example output

```
site-install
└── site-1-sno
    ├── site-1_agentclusterinstall_example-sno.yaml
    ├── site-1-sno_baremetalhost_example-node1.example.com.yaml
    ├── site-1-sno_clusterdeployment_example-sno.yaml
    ├── site-1-sno_configmap_example-sno.yaml
    ├── site-1-sno_infraenv_example-sno.yaml
    ├── site-1-sno_klusterletaddonconfig_example-sno.yaml
    ├── site-1-sno_machineconfig_02-master-workload-partitioning.yaml
├── site-1-sno_machineconfig_predefined-extra-manifests-master.yaml
├── site-1-sno_machineconfig_predefined-extra-manifests-worker.yaml
├── site-1-sno_managedcluster_example-sno.yaml
├── site-1-sno_namespace_example-sno.yaml
└── site-1-sno_nmstateconfig_example-node1.example.com.yaml
```

6. Optional: Generate just the day-0 **MachineConfig** installation CRs for a particular cluster type by processing the reference **SiteConfig** CR with the **-E** option. For example, run the following commands:

   a. Create an output folder for the **MachineConfig** CRs:

   ```
   $ mkdir -p ./site-machineconfig
   ```

   b. Generate the **MachineConfig** installation CRs:

   ```
   $ podman run -it --rm -v `pwd`/out/argocd/example/siteconfig:/resources:Z -v `pwd`/site-machineconfig:/output:Z,U registry.redhat.io/openshift4/ztp-site-generate-rhel8:v4.12.1 generator install -E site-1-sno.yaml /output
   ```

   ### Example output

   ```
   site-machineconfig
    └── site-1-sno
        ├── site-1-sno_machineconfig_02-master-workload-partitioning.yaml
        ├── site-1-sno_machineconfig_predefined-extra-manifests-master.yaml
        └── site-1-sno_machineconfig_predefined-extra-manifests-worker.yaml
   ```

7. Generate and export the day-2 configuration CRs using the reference **PolicyGenTemplate** CRs from the previous step. Run the following commands:

   a. Create an output folder for the day-2 CRs:

   ```
   $ mkdir -p ./ref
   ```

   b. Generate and export the day-2 configuration CRs:

   ```
   $ podman run -it --rm -v `pwd`/out/argocd/example/policygentemplates:/resources:Z -v `pwd`/ref:/output:Z,U registry.redhat.io/openshift4/ztp-site-generate-rhel8:v4.12.1 generator config -N . /output
   ```

   The command generates example group and site-specific **PolicyGenTemplate** CRs for single-node OpenShift, three-node clusters, and standard clusters in the **./ref** folder.

   ### Example output

   ```
   ref
    └── customResource
        ├── common
        ├── example-multinode-site
        ├── example-sno
        ├── group-du-3node
        ├── group-du-3node-validator
        │   └── Multiple-validatorCRs
   ```

```
    ├── group-du-sno
    ├── group-du-sno-validator
    ├── group-du-standard
    └── group-du-standard-validator
        └── Multiple-validatorCRs
```

8. Use the generated CRs as the basis for the CRs that you use to install the cluster. You apply the installation CRs to the hub cluster as described in "Installing a single managed cluster". The configuration CRs can be applied to the cluster after cluster installation is complete.

Additional resources

- Workload partitioning
- BMC addressing

17.5.2. Creating the managed bare-metal host secrets

Add the required **Secret** custom resources (CRs) for the managed bare-metal host to the hub cluster. You need a secret for the ZTP pipeline to access the Baseboard Management Controller (BMC) and a secret for the assisted installer service to pull cluster installation images from the registry.

> **NOTE**
>
> The secrets are referenced from the **SiteConfig** CR by name. The namespace must match the **SiteConfig** namespace.

Procedure

1. Create a YAML secret file containing credentials for the host Baseboard Management Controller (BMC) and a pull secret required for installing OpenShift and all add-on cluster Operators:

   a. Save the following YAML as the file **example-sno-secret.yaml**:

```
apiVersion: v1
kind: Secret
metadata:
  name: example-sno-bmc-secret
  namespace: example-sno 1
data: 2
  password: <base64_password>
  username: <base64_username>
type: Opaque
---
apiVersion: v1
kind: Secret
metadata:
  name: pull-secret
  namespace: example-sno  3
data:
  .dockerconfigjson: <pull_secret> 4
type: kubernetes.io/dockerconfigjson
```

1 Must match the namespace configured in the related **SiteConfig** CR

2 Base64-encoded values for **password** and **username**

3 Must match the namespace configured in the related **SiteConfig** CR

4 Base64-encoded pull secret

2. Add the relative path to **example-sno-secret.yaml** to the **kustomization.yaml** file that you use to install the cluster.

17.5.3. Configuring Discovery ISO kernel arguments for manual installations using GitOps ZTP

The GitOps ZTP workflow uses the Discovery ISO as part of the OpenShift Container Platform installation process on managed bare-metal hosts. You can edit the **InfraEnv** resource to specify kernel arguments for the Discovery ISO. This is useful for cluster installations with specific environmental requirements. For example, configure the **rd.net.timeout.carrier** kernel argument for the Discovery ISO to facilitate static networking for the cluster or to receive a DHCP address before downloading the root file system during installation.

NOTE

In OpenShift Container Platform 4.12, you can only add kernel arguments. You can not replace or delete kernel arguments.

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 manually generated the installation and configuration custom resources (CRs).

Procedure

1. Edit the **spec.kernelArguments** specification in the **InfraEnv** CR to configure kernel arguments:

```
apiVersion: agent-install.openshift.io/v1beta1
kind: InfraEnv
metadata:
  name: <cluster_name>
  namespace: <cluster_name>
spec:
  kernelArguments:
    - operation: append 1
      value: audit=0 2
    - operation: append
      value: trace=1
  clusterRef:
    name: <cluster_name>
    namespace: <cluster_name>
  pullSecretRef:
    name: pull-secret
```

1. Specify the append operation to add a kernel argument.

2. Specify the kernel argument you want to configure. This example configures the audit kernel argument and the trace kernel argument.

NOTE

The **SiteConfig** CR generates the **InfraEnv** resource as part of the day-0 installation CRs.

Verification

To verify that the kernel arguments are applied, after the Discovery image verifies that OpenShift Container Platform is ready for installation, you can SSH to the target host before the installation process begins. At that point, you can view the kernel arguments for the Discovery ISO in the **/proc/cmdline** file.

1. Begin an SSH session with the target host:

   ```
   $ ssh -i /path/to/privatekey core@<host_name>
   ```

2. View the system’s kernel arguments by using the following command:

   ```
   $ cat /proc/cmdline
   ```

17.5.4. Installing a single managed cluster

You can manually deploy a single managed cluster using the assisted service and Red Hat Advanced Cluster Management (RHACM).

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 the baseboard management controller (BMC) **Secret** and the image pull-secret **Secret** custom resources (CRs). See "Creating the managed bare-metal host secrets" for details.
- Your target bare-metal host meets the networking and hardware requirements for managed clusters.

Procedure

1. Create a **ClusterImageSet** for each specific cluster version to be deployed, for example **clusterImageSet-4.12.yaml**. A **ClusterImageSet** has the following format:

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

```
kind: ClusterImageSet
metadata:
  name: openshift-4.12.0 1
spec:
   releaseImage: quay.io/openshift-release-dev/ocp-release:4.12.0-x86_64 2
```

1 The descriptive version that you want to deploy.

2 Specifies the **releaseImage** to deploy and determines the operating system image version. The discovery ISO is based on the image version as set by **releaseImage**, or the latest version if the exact version is unavailable.

2. Apply the **clusterImageSet** CR:

```
$ oc apply -f clusterImageSet-4.12.yaml
```

3. Create the **Namespace** CR in the **cluster-namespace.yaml** file:

```
apiVersion: v1
kind: Namespace
metadata:
     name: <cluster_name> 1
     labels:
        name: <cluster_name> 2
```

1 2 The name of the managed cluster to provision.

4. Apply the **Namespace** CR by running the following command:

```
$ oc apply -f cluster-namespace.yaml
```

5. Apply the generated day-0 CRs that you extracted from the **ztp-site-generate** container and customized to meet your requirements:

```
$ oc apply -R ./site-install/site-sno-1
```

Additional resources

- Connectivity prerequisites for managed cluster networks

17.5.5. Monitoring the managed cluster installation 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>
   ```

6. Check the status of the managed cluster's add-on services:

   ```
   $ oc get managedclusteraddon -n <cluster_name>
   ```

7. 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
   ```

17.5.6. Troubleshooting the managed cluster

Use this procedure to diagnose any installation issues that might occur with the managed cluster.

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.

17.5.7. RHACM generated cluster installation CRs reference

Red Hat Advanced Cluster Management (RHACM) supports deploying OpenShift Container Platform on single-node clusters, three-node clusters, and standard clusters with a specific set of installation custom resources (CRs) that you generate using **SiteConfig** CRs for each site.

NOTE

Every managed cluster has its own namespace, and all of the installation CRs except for **ManagedCluster** and **ClusterImageSet** are under that namespace. **ManagedCluster** and **ClusterImageSet** are cluster-scoped, not namespace-scoped. The namespace and the CR names match the cluster name.

The following table lists the installation CRs that are automatically applied by the RHACM assisted service when it installs clusters using the **SiteConfig** CRs that you configure.

Table 17.5. Cluster installation CRs generated by RHACM

CR	Description	Usage
BareMetalHost	Contains the connection information for the Baseboard Management Controller (BMC) of the target bare-metal host.	Provides access to the BMC to load and start the discovery image on the target server by using the Redfish protocol.
InfraEnv	Contains information for installing OpenShift Container Platform on the target bare-metal host.	Used with **ClusterDeployment** to generate the discovery ISO for the managed cluster.
AgentClusterInstall	Specifies details of the managed cluster configuration such as networking and the number of control plane nodes. Displays the cluster **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** CR to use.	Used with **InfraEnv** to generate the discovery ISO for the managed cluster.
NMStateConfig	Provides network configuration information such as **MAC** address to **IP** mapping, DNS server, default route, and other network settings.	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 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 the **ManagedCluster** resource.	Tells the hub which addon services to deploy to the **ManagedCluster** resource.
Namespace	Logical space for **ManagedCluster** resources existing on the hub. Unique per site.	Propagates resources to the **ManagedCluster**.

CR	Description	Usage
Secret	Two CRs are created: **BMC Secret** and **Image Pull Secret**.	● **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.
ClusterImageSet	Contains OpenShift Container Platform image information such as the repository and image name.	Passed into resources to provide OpenShift Container Platform images.

17.6. RECOMMENDED SINGLE-NODE OPENSHIFT CLUSTER CONFIGURATION FOR VDU APPLICATION WORKLOADS

Use the following reference information to understand the single-node OpenShift configurations required to deploy virtual distributed unit (vDU) applications in the cluster. Configurations include cluster optimizations for high performance workloads, enabling workload partitioning, and minimizing the number of reboots required postinstallation.

Additional resources

- To deploy a single cluster by hand, see Manually installing a single-node OpenShift cluster with ZTP.
- To deploy a fleet of clusters using GitOps zero touch provisioning (ZTP), see Deploying far edge sites with ZTP.

17.6.1. Running low latency applications on OpenShift Container Platform

OpenShift Container Platform enables low latency processing for applications running on commercial off-the-shelf (COTS) hardware by using several 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-aware topology management

Aligns memory and huge pages with CPU and PCI devices to pin guaranteed container memory and huge pages to the non-uniform memory access (NUMA) node. Pod resources for all Quality of Service (QoS) classes stay on the same 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.

17.6.2. Recommended cluster host requirements for vDU application workloads

Running vDU application workloads requires a bare-metal host with sufficient resources to run OpenShift Container Platform services and production workloads.

Table 17.6. Minimum resource requirements

Profile	vCPU	Memory	Storage
Minimum	4 to 8 vCPU cores	32GB of RAM	120GB

NOTE

One vCPU is equivalent to one physical core when simultaneous multithreading (SMT), or Hyper-Threading, is not enabled. When enabled, use the following formula to calculate the corresponding ratio:

- (threads per core × cores) × sockets = vCPUs

IMPORTANT

The server must have a Baseboard Management Controller (BMC) when booting with virtual media.

17.6.3. Configuring host firmware for low latency and high performance

Bare-metal hosts require the firmware to be configured before the host can be provisioned. The firmware configuration is dependent on the specific hardware 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 firmware configuration for your hardware. The following table describes a representative firmware 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 firmware configuration depends on your specific hardware and network requirements. The following sample configuration is for illustrative purposes only.

   Table 17.7. Sample firmware configuration for an Intel Xeon Skylake or Cascade Lake server

Firmware 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
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 firmware for the host. These settings are relevant to bare-metal environments.

17.6.4. Connectivity prerequisites for managed cluster networks

Before you can install and provision a managed cluster with the zero touch provisioning (ZTP) GitOps pipeline, the managed cluster host must meet the following networking prerequisites:

- There must be bi-directional connectivity between the ZTP GitOps container in the hub cluster and the Baseboard Management Controller (BMC) of the target bare-metal host.

- The managed cluster must be able to resolve and reach the API hostname of the hub hostname and *.apps hostname. Here is an example of the API hostname of the hub and *.apps hostname:

  - **api.hub-cluster.internal.domain.com**

  - **console-openshift-console.apps.hub-cluster.internal.domain.com**

- The hub cluster must be able to resolve and reach the API and *.apps hostname of the managed cluster. Here is an example of the API hostname of the managed cluster and *.apps hostname:

- **api.sno-managed-cluster-1.internal.domain.com**
- **console-openshift-console.apps.sno-managed-cluster-1.internal.domain.com**

17.6.5. Workload partitioning in single-node OpenShift with GitOps ZTP

Workload partitioning configures OpenShift Container Platform services, cluster management workloads, and infrastructure pods to run on a reserved number of host CPUs.

To configure workload partitioning with GitOps ZTP, you specify cluster management CPU resources with the **cpuset** field of the **SiteConfig** custom resource (CR) and the **reserved** field of the group **PolicyGenTemplate** CR. The GitOps ZTP pipeline uses these values to populate the required fields in the workload partitioning **MachineConfig** CR (**cpuset**) and the **PerformanceProfile** CR (**reserved**) that configure the single-node OpenShift cluster.

NOTE

For maximum performance, ensure that the **reserved** and **isolated** CPU sets do not share CPU cores across NUMA zones.

- The workload partitioning **MachineConfig** CR pins the OpenShift Container Platform infrastructure pods to a defined **cpuset** configuration.
- The **PerformanceProfile** CR pins the systemd services to the reserved CPUs.

IMPORTANT

The value for the **reserved** field specified in the **PerformanceProfile** CR must match the **cpuset** field in the workload partitioning **MachineConfig** CR.

Additional resources

- For the recommended single-node OpenShift workload partitioning configuration, see Workload partitioning.

17.6.6. Recommended installation-time cluster configurations

The ZTP pipeline applies the following custom resources (CRs) during cluster installation. These configuration CRs ensure that the cluster meets the feature and performance requirements necessary for running a vDU application.

NOTE

When using the ZTP GitOps plugin and **SiteConfig** CRs for cluster deployment, the following **MachineConfig** CRs are included by default.

Use the **SiteConfig extraManifests** filter to alter the CRs that are included by default. For more information, see Advanced managed cluster configuration with SiteConfig CRs .

17.6.6.1. Workload partitioning

Single-node OpenShift clusters that run DU workloads require workload partitioning. This limits the cores allowed to run platform services, maximizing the CPU core for application payloads.

> NOTE
>
> Workload partitioning can only be enabled during cluster installation. You cannot disable workload partitioning postinstallation. However, you can reconfigure workload partitioning by updating the **cpu** value that you define in the performance profile, and in the related **MachineConfig** custom resource (CR).

- The base64-encoded CR that enables workload partitioning contains the CPU set that the management workloads are constrained to. Encode host-specific values for **crio.conf** and **kubelet.conf** in base64. Adjust the content to match the CPU set that is specified in the cluster performance profile. It must match the number of cores in the cluster host.

  Recommended workload partitioning configuration

```
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,W2NyaW8ucnVudGltZS53b3JrbG9hZHMubWFuYWdlbWVudF0KYWN0aXZhdGlvbl
9hbm5vdGF0aW9uID0gInRhcmdldC53b3JrbG9hZC5vcGVuc2hpZnQuaW8vbWFuYWdlbWVu
dCIKYW5ub3RhdGlvbl9wcmVmaXggPSAicmVzb3VyY2VzLndvcmtsb2FkLm9wZW5zaGlmdC5
pbyIKcmVzb3VyY2VzID0geyAiY3B1c2hhcmVzIiA9IDAsICJjcHVzZXQiID0gIjAtMSw1Mi01MyIgf
Qo=
        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,ewogICJtYW5hZ2VtZW50IjogewogICAgImNwdXNldCI6ICIwLTEsNTItNTMiCiAgfQp
9Cg==
        mode: 420
        overwrite: true
        path: /etc/kubernetes/openshift-workload-pinning
        user:
          name: root
```

- When configured in the cluster host, 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"
resources = { "cpushares" = 0, "cpuset" = "0-1,52-53" } 1
```

1 The **cpuset** value varies based on the installation. If Hyper-Threading is enabled, specify both threads for each core. The **cpuset** value must match the reserved CPUs that you define in the **spec.cpu.reserved** field in the performance profile.

- When configured in the cluster, 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 **cpuset** value in **/etc/crio/crio.conf.d/01-workload-partitioning**.

Verification

Check that the applications and cluster system CPU pinning is correct. Run the following commands:

1. Open a remote shell connection to the managed cluster:

```
$ oc debug node/example-sno-1
```

2. Check that the OpenShift infrastructure applications CPU pinning is correct:

```
sh-4.4# pgrep ovn | while read i; do taskset -cp $i; done
```

Example output

```
pid 8481's current affinity list: 0-1,52-53
pid 8726's current affinity list: 0-1,52-53
pid 9088's current affinity list: 0-1,52-53
pid 9945's current affinity list: 0-1,52-53
pid 10387's current affinity list: 0-1,52-53
pid 12123's current affinity list: 0-1,52-53
pid 13313's current affinity list: 0-1,52-53
```

3. Check that the system applications CPU pinning is correct:

```
sh-4.4# pgrep systemd | while read i; do taskset -cp $i; done
```

Example output

```
pid 1's current affinity list: 0-1,52-53
pid 938's current affinity list: 0-1,52-53
pid 962's current affinity list: 0-1,52-53
pid 1197's current affinity list: 0-1,52-53
```

17.6.6.2. Reduced platform management footprint

To reduce the overall management footprint of the platform, a **MachineConfig** custom resource (CR) is required that places all Kubernetes-specific mount points in a new namespace separate from the host operating system. The following base64-encoded example **MachineConfig** CR illustrates this configuration.

Recommended container mount namespace configuration

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: container-mount-namespace-and-kubelet-conf-master
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
      files:
      - contents:
          source: data:text/plain;charset=utf-
8;base64,IyEvYmluL2Jhc2gKCmRlYnVnKCkgewogIGVjaG8gJEAgPiYyCn0KCnVzYWdlKCkgewogIGVj
aG8gVXNhZ2U6ICQoYmFzZW5hbWUgJDApIFVOSVQgW2VudmZpbGUgW3Zhcm5hbWVdXQogIGV
jaG8KICBlY2hvIEV4dHJhY3QgdGhlIGNvbnRlbnRzIG9mIHRoZSBmaXJzdCBFeGVjU3RhcnQgc3Rhbn
phIGZyb20gdGhlIGdpdmVuIHN5c3RlbWQgdW5pdCBhbmQgcmV0dXJuIGl0IHRvIHN0ZG91dAogIGVj
aG8KICBlY2hvICJJZiAnZW52ZmlsZScgaXMgcHJvdmlkZWQsIHB1dCBpdCBpbiB0aGVyZSBpbnN0ZW
FkLCBhcyBhbiBlbnZpcm9ubWVudCB2YXJpYWJsZSBuYW1lZCAndmFybmFtZSciCiAgZWNobyAiRGV
mYXVsdCAndmFybmFtZScgaXMgRVhFQ1NUQVJUIGlmIG5vdCBzcGVjaWZpZWQiCiAgZXhpdCAxC
n0KClVOSVQ9JDEKRU5WRklMRT0kMgpWQVJOQU1FPSQzCmlmIFtbIC16ICRVTklUIHx8ICRVTklUI
D09ICItLWhlbHAiIHx8ICRVTklUID09ICItaCIgXV07IHRoZW4KICB1c2FnZQpmaQpkZWJ1ZyAiRXh0cm
FjdGluZyBFeGVjU3RhcnQgZnJvbSAkVU5JVCIKRklMRT0kKHN5c3RlbWN0bCBjYXQgJFVOSVQgfCB
oZWFkIC1uIDEpCkZJTEU9JHtGSUxFI1wjIH0KaWYgW1sgISAtZiAkRklMRSBdXTsgdGhlbgogIGRlYnV
nICJGYWlsZWQgdG8gZmluZCByb290IGZpbGUgZm9yIHVuaXQgJFVOSVQgKCRGSUxFKSIKICBle
Gl0CmZpCmRlYnVnICJTZXJ2aWNlIGRlZmluaXRpb24gaXMgaW4gJEZJTEUiCkVYRUNTVEFSVD0k
KHNlZCAtbiAtZSAnL15FeGVjU3RhcnQ9LipcXCQvLC9bXlxcXSQvIHsgcy9eRXhlY1N0YXJ0PS8vOyBw
IH0nIC1lICcvXkV4ZWNTdGFydD0uKlteXFxdJC8geyBzL15FeGVjU3RhcnQ9Ly87IHAgfScgJEZJTEUp
CgppZiBbWyAkRU5WRklMRSBdXTsgdGhlbgogIFZBUk5BTUU9JHtWQVJOQU1FOi1FWEVDU1RBUl
R9CiAgZWNobyAiJHtWQVJOQU1FfT0ke0VYRUNTVEFSVH0iID4gJEVOVkZJTEUKZWxzZQogIGVja
G8gJEVYRUNTVEFSVApmaQo=
        mode: 493
        path: /usr/local/bin/extractExecStart
      - contents:
          source: data:text/plain;charset=utf-
8;base64,IyEvYmluL2Jhc2gKbnNlbnRlciAtLW1vdW50PS9ydW4vY29udGFpbmVyLW1vdW50LW5hbWV
zcGFjZS9tbnQgIiRAIgo=
        mode: 493
        path: /usr/local/bin/nsenterCmns
    systemd:
      units:
      - contents: |
          [Unit]
          Description=Manages a mount namespace that both kubelet and crio can use to share their
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="OPENSHIFT_MAX_HOUSEKEEPING_INTERVAL_DURATION=60s"
            Environment="OPENSHIFT_EVICTION_MONITORING_PERIOD_DURATION=30s"
          name: 30-kubelet-interval-tuning.conf
        name: kubelet.service
```

17.6.6.3. SCTP

Stream Control Transmission Protocol (SCTP) is a key protocol used in RAN applications. This **MachineConfig** object adds the SCTP kernel module to the node to enable this protocol.

Recommended SCTP configuration

```
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
```

17.6.6.4. Accelerated container startup

The following **MachineConfig** CR configures core OpenShift processes and containers to use all available CPU cores during system startup and shutdown. This accelerates the system recovery during initial boot and reboots.

Recommended accelerated container startup configuration

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: 04-accelerated-container-startup-master
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
      files:
      - contents:
          source: data:text/plain;charset=utf-
8;base64,IyEvYmluL2Jhc2gKIwojIFRlbXBvcmFyaWx5IHJlc2V0IHRoZSBjb3JlIHN5c3RlbSBwcm9jZXNz
ZXMncyBDUFUgYWZmaW5pdHkgdG8gYmUgdW5yZXN0cmljdGVkIHRvIGFjY2VsZXJhdGUgc3Rhcn
R1cCBhbmQgc2h1dGRvd24KIwojIFRoZSBkZWZhdWx0cyBiZWxvdyBjYW4gYmUgb3ZlcnJpZGRlbiB2a
WEgZW52aXJvbm1lbnQgdmFyaWFibGVzCiMKCiMgVGhlIGRlZmF1bHQgc2V0IG9mIGNyaXRpY2FsI
HByb2Nlc3NlcyB3aG9zZSBhZmZpbml0eSBzaG91bGQgYmUgdGVtcG9yYXJpbHkgdW5ib3VuZDoKQ
1JJVElDQUxfUFJPQ0VTU0VTPSR7Q1JJVElDQUxfUFJPQ0VTU0VTOi0iY3JpbyBrdWJlbGV0IE5ldHd
vcmtNYW5hZ2VyIGNvbm1vbiBkYnVzIn0KCiMgRGVmYXVsdCB3YWl0IHRpbWUgaXMgNjAwcyA9IDE
wbToKTUFYSU1VTV9XQUlUX1RJTUU9JHtNQVhJTVVNX1dBSVRfVElNRTotNjAwfQoKIyBEZWZhd
Wx0IHN0ZWFkeS1zdGF0ZSB0aHJlc2hvbGQgPSAyJQojIEFsbG93ZWQgdmFsdWVzOgojICA0ICAtIG
Fic29sdXRlIHBvZCBjb3VudCAoKy8tKQojICA0JSAtIHBlcmNlbnQgY2hhbmdlICgrLy0pCiMgIC0xIC0gZG
lzYWJsZSB0aGUgc3RlYWR5LXN0YXRlIGNoZWNrClNURUFEWV9TVEFURV9USFJFU0hPTEQ9JHt
TVEVBRFlfU1RBVEVfVEhSRVNIT0xEOi0yJX0KCiMgRGVmYXVsdCBzdGVhZHktc3RhdGUgd2luZG9
```

```
3ID0gNjBzCiMgSWYgdGhlIHJ1bm5pbmcgcG9kIGNvdW50IHN0YXlzIHdpdGhpbiB0aGUgZ2l2ZW4gdG
hyZXNob2xkIGZvciB0aGlzIHRpbWUKIyBwZXJpb2QsIHJldHVybiBDUFUgdXRpbGl6YXRpb24gdG8gb
m9ybWFsIGJlZm9yZSB0aGUgbWF4aW11bSB3YWl0IHRpbWUgaGFzCiMgZXhwaXJlcwpTVEVBRFlf
U1RBVEVfV0lORE9XPSR7U1RFQURZX1NUQVRFX1dJTkRPVzotNjB9CgojIERlZmF1bHQgc3RlYW
R5LXN0YXRlIGFsbG93cyBhbnkgcG9kIGNvdW50IHRvIGJlICJzdGVhZHkgc3RhdGUiCiMgSW5jcmVhc
2luZyB0aGlzIHdpbGwgc2tpcCBhbnkgc3RlYWR5LXN0YXRlIGNoZWNrcyB1bnRpbCB0aGUgY291bnQ
gcmlzZXMgYWJvdmUKIyB0aGlzIG51bWJlciB0byBhdm9pZCBmYWxzZSBwb3NpdGl2ZXMgaWYgdGhl
cmUgYXJlIHNvbWUgcGVyaW9kcyB3aGVyZSB0aGUKIyBjb3VudCBkb2Vzbid0IGluY3JlYXNlIGJ1dCB3
ZSBrbm93IHdlIGNhbid0IGJlIGF0IHN0ZWFkeS1zdGF0ZSB5ZXQuClNURUFEWV9TVEFURV9NSU5J
TVVNPSR7U1RFQURZX1NUQVRFX01JTklNVU06LTB9CgojIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjI
yMjIyMjIyMjIyMjIyMjIyMjCgpLVUJFTEVUX0NQVV9TVEFURT0vdmFyL2xpYi9rdWJlbGV0L2Nw
dV9tYW5hZ2VyX3N0YXRlCkZVTExfQ1BVX1NUQVRFPS9zeXMvZnMvY2dyb3VwL2NwdXNldC9jcHVz
ZXQuY3B1cwpLVUJFTEVUX0NPTkY9L2V0Yy9rdWJlcm5ldGVzL2t1YmVsZXQuY29uZgp1bnJlc3Rya
WN0ZWRDcHVzZXQoKSB7CiAgbG9jYWwgY3B1cwogIGlmIFtbIC1lICRLVUJFTEVUX0NQVV9TVEF
URSBdXTsgdGhlbgogICAgY3B1cz0kKGpxIC1yICcuZGVmYXVsdENwdVNldCcgPCRLVUJFTEVUX0N
QVV9TVEFURSkKICAgIGlmIFtbIC1uICIke2NwdXN9IiAmJiAtZSAke0tVQkVMRVRfQ09ORn0gXV07IH
RoZW4KICAgICAgcmVzZXJ2ZWRfY3B1cz0kKGpxIC1yICcucmVzZXJ2ZWRTeXN0ZW1DUFVzJyA8L2
V0Yy9rdWJlcm5ldGVzL2t1YmVsZXQuY29uZikKICAgICAgaWYgW1sgLW4gIiR7cmVzZXJ2ZWRfY3B1c
30iIF1dOyB0aGVuCiAgICAgICAgIyBVc2UgdGFza3NldCB0byBtZXJnZSB0aGUgdHdvIGNwdXNldHMK
ICAgICAgICBjcHVzPSQodGFza3NldCAtYyAiJHtyZXNlcnZlZF9jcHVzfSwke2NwdXN9IiBncmVwIC1pIE
NwdXNfYWxsb3dlZF9saXN0IC9wcm9jL3NlbGYvc3RhdHVzIHwgYXdrICd7cHJpbnQgJDJ9JykKICAgIC
AgZmkKICAgIGZpCiAgZmkKICBpZiBbWyAteiAkY3B1cyBdXTsgdGhlbgogICAgIyBmYWxsIGJhY2sgdG
8gdXNpbmcgYWxsIGNwdXMgaWYgdGhlIGt1YmVsZXQgc3RhdGUgaXMgbm90IGNvbmZpZ3VyZWQ
geWV0CiAgICBbWyAtZSAkRlVMTF9DUFVfU1RBVEUgXV0gfHwgcmV0dXJuIDEKICAgIGNwdXM9JC
g8JEZVTExfQ1BVX1NUQVRFKQogIGZpCiAgZWNobyAkY3B1cwp9CgpyZXN0cmljdGVkQ3B1c2V0K
CkgewogIGZvciBhcmcgaW4gJCg8L3Byb2MvY21kbGluZSk7IGRvCiAgICBpZiBbWyAkYXJnID1+IF5zeX
N0ZW1kLmNwdV9hZmZpbml0eT0gXV07IHRoZW4KICAgICAgZWNobyAke2FyZyMqPX0KICAgICAgc
mV0dXJuIDAKICAgIGZpCiAgZG9uZQogIHJldHVybiAxCn0KCnJlc2V0QWZmaW5pdHkoKSB7CiAgbG
9jYWwgY3B1c2V0PSIkMSIKICBsb2NhbCBmYWlsY291bnQ9MAogIGxvY2FsIHN1Y2Nlc3Njb3VudD0w
CiAgbG9nZ2VyICJSZWNvdmVyeTogU2V0dGluZyBDUFUgYWZmaW5pdHkgZm9yIGNyaXRpY2FsIH
Byb2Nlc3NlcyBcIiRDUklUSUNBTF9QUk9DRVNTRVNcIiB0byAkY3B1c2V0IgogIGZvciBwcm9jIGluICRD
UklUSUNBTF9QUk9DRVNTRVM7IGRvCiAgICBsb2NhbCBwaWRzPSIkKHBncmVwICRwcm9jKSIKIC
AgIGZvciBwaWQgaW4gJHBpZHM7IGRvCiAgICAgIGxvY2FsIHRhc2tzZXRPdXRwdXQKICAgICAgdG
Fza3NldE91dHB1dD0iJCh0YXNrc2V0IC1hcGMgIiRjcHVzZXQiICRwaWQgMj4mMSkiCiAgICAgIGlmIFt
bICQ/IC1uZSAwIF1dOyB0aGVuCiAgICAgICAgZWNobyAiRVJST1I6ICR0YXNrc2V0T3V0cHV0IgogIC
AgICAgICgoZmFpbGNvdW50KyspKQogICAgICBlbHNlCiAgICAgICAgKChzdWNjZXNzY291bnQrKykp
CiAgICAgIGZpCiAgICBkb25lCiAgZG9uZQoKICBsb2dnZXIgIlJlY292ZXJ5OiBSZS1hZmZpbmVkICRzd
WNjZXNzY291bnQgcGlkcyBzdWNjZXNzZnVsbHkiCiAgaWYgW1sgJGZhaWxjb3VudCAtZ3QgMCBdXT
sgdGhlbgogICAgbG9nZ2VyICJSZWNvdmVyeTogRmFpbGVkIHRvIHJlLWFmZmluZSAkZmFpbGNvdW
50IHByb2Nlc3NlcyIKICAgIHJldHVybiAxCiAgZmkKfQoKc2V0VW5yZXN0cmljdGVkKCkgewogIGxvZ2dlci
AiUmVjb3Zlcnk6IFNldHRpbmcgY3JpdGljYWwgc3lzdGVtIHByb2Nlc3NlcyB0byBoYXZlIHVucmVzdHJpY
RlZCBDUFUgYWNjZXNzIgogIHJlc2V0QWZmaW5pdHkgIiQodW5yZXN0cmljdGVkQ3B1c2V0KSIKfQo
Kc2V0UmVzdHJpY3RlZCgpIHsKICBsb2dnZXIgIlJlY292ZXJ5OiBSZXNldHRpbmcgY3JpdGljYWwgc3lzd
GVtIHByb2Nlc3NlcyBiYWNrIHRvIG5vcm1hbGx5IHJlc3RyaWN0ZWQgYWNjZXNzIgogIHJlc2V0QWZm
aW5pdHkgIiQocmVzdHJpY3RlZENwdXNldCkiCn0KCmN1cnJlbnRBZmZpbml0eSgpIHsKICBsb2NhbC
BwaWQ9IiQxIgogIHRhc2tzZXQgLXBjICRwaWQgfCBhd2sgLUYnOiAnICd7cHJpbnQgJDJ9Jwp9Cgp3a
XRoaW4oKSB7CiAgbG9jYWwgbGFzdD0kMSBjdXJyZW50PSQyIHRocmVzaG9sZD0kMwogIGxvY2FsI
GRlbHRhPTAgcGNoYW5nZQogIGRlbHRhPSQoKCBjdXJyZW50IC0gbGFzdCApKQogIGlmIFtbICRjd
XJyZW50IC1lcSAkbGFzdCBdXTsgdGhlbgogICAgcGNoYW5nZT0wCiAgZWxpZiBbWyAkbGFzdCAtZX
EgMCBdXTsgdGhlbgogICAgcGNoYW5nZT0xMDAwMDAwCiAgZWxzZQogICAgcGNoYW5nZT0kKCg
gKCAkZGVsdGEgKiAxMDApIC8gbGFzdCApKQogIGZpCiAgZWNobyAtbiAibGFzdDokbGFzdCBjdXJy
ZW50OiRjdXJyZW50IGRlbHRhOiRkZWx0YSBwY2hhbmdlOiR7cGNoYW5nZX0lOiAiCiAgbG9jYWwgY
WJzb2x1dGUgbGltaXQKICBjYXNlICR0aHJlc2hvbGQgaW4KICAgIColKQogICAgICBhYnNvbHV0ZT0k
e3BjaGFuZ2UjIy19ICMgYWJzb2x1dGUgdmFsdWUKICAgICAgbGltaXQ9JHt0aHJlc2hvbGQlJSV9CiAgI
CAgIDs7CiAgICAqKQogICAgICBhYnNvbHV0ZT0ke2RlbHRhIyMtfSAjIGFic29sdXRlIHZhbHVlCiAgICAg
IGxpbWl0PSR0aHJlc2hvbGQKICAgICAgOzsKICBlc2FjCiAgaWYgW1sgJGFic29sdXRlIC1sZSAkbGltaX
```

```
QgXV07IHRoZW4KICAgIGVjaG8gIndpdGhpbiAoKy8tKSR0aHJlc2hvbGQiCiAgICByZXR1cm4gMAogI
GVsc2UKICAgIGVjaG8gIm91dHNpZGUgKCsvLSkkdGhyZXNob2xkIgogICAgcmV0dXJuIDEKICBmaQ
p9CgpzdGVhZHlzdGF0ZSgpIHsKICBsb2NhbCBsYXN0PSQxIGN1cnJlbnQ9JDIKICBpZiBbWyAkbGFz
dCAtbHQgJFNURUFEWV9TVEFURV9NSU5JTVVNIF1dOyB0aGVuCiAgICBlY2hvICJsYXN0OiRsYXN
0IGN1cnJlbnQ6JGN1cnJlbnQgV2FpdGluZyB0byByZWFjaCAkU1RFQURZX1NUQVRFX01JTklNVU0g
YmVmb3JlIGNoZWNraW5nIGZvciBzdGVhZHktc3RhdGUiCiAgICByZXR1cm4gMQogIGZpCiAgd2l0aGl
uICRsYXN0ICRjdXJyZW50ICRTVEVBRFlfU1RBVEVfVEhSRVNIT0xECn0KCndhaXRGb3JSZWFkeSg
pIHsKICBsb2dnZXIgIlJlY292ZXJ5OiBXYWl0aW5nICR7TUFYSU1VTV9XQUlUX1RJTUV9cyBmb3IgdG
hlIGluaXRpYWxpemF0aW9uIHRvIGNvbXBsZXRlIgogIGxvY2FsIGxhc3RTeXN0ZW1kQ3B1c2V0PSIkK
GN1cnJlbnRBZmZpbml0eSAxKSIKICBsb2NhbCBsYXN0RGVzaXJlZENwdXNldD0iJCh1bnJlc3RyaWN
0ZWRDcHVzZXQpIgogIGxvY2FsIHQ9MCBzPTEwCiAgbG9jYWwgbGFzdENjb3VudD0wIGNjb3VudD0
wIHN0ZWFkeVN0YXRlVGltZT0wCiAgd2hpbGUgW1sgJHQgLWx0ICRNQVhJTVVNX1dBSVRfVElNRS
BdXTsgZG8KICAgIHNsZWVwICRzCiAgICAoKHQgKz0gcykpCiAgICAjIFJlLWNoZWNrIHRoZSBjdXJyZ
W50IGFmZmluaXR5IG9mIHN5c3RlbWQsIGluIGNhc2Ugc29tZSBvdGhlciBwcm9jZXNzIGhhcyBjaGFuZ
2VkIGl0CiAgICBsb2NhbCBzeXN0ZW1kQ3B1c2V0PSIkKGN1cnJlbnRBZmZpbml0eSAxKSIKICAgICMg
UmUtY2hlY2sgdGhlIHVucmVzdHJpY3RlZCBDcHVzZXQsIGFzIHRoZSBhbGxvd2VkIHNldCBvZiB1bnJl
c2VydmVkIGNvcmVzIG1heSBjaGFuZ2UgYXMgcG9kcyBhcmUgYXNzaWduZWQgdG8gY29yZXMKICA
gIGxvY2FsIGRlc2lyZWRDcHVzZXQ9IiQodW5yZXN0cmljdGVkQ3B1c2V0KSIKICAgIGlmIFtbICRzeXN0
ZW1kQ3B1c2V0ICE9ICRsYXN0U3lzdGVtZENwdXNldCB8fCAkbGFzdERlc2lyZWRDcHVzZXQgIT0gJ
GRlc2lyZWRDcHVzZXQgXV07IHRoZW4KICAgICAgcmVzZXRBZmZpbml0eSAiJGRlc2lyZWRDcHVzZ
XQiCiAgICAgIGxhc3RTeXN0ZW1kQ3B1c2V0PSIkKGN1cnJlbnRBZmZpbml0eSAxKSIKICAgICAgbGF
zdERlc2lyZWRDcHVzZXQ9IiRkZXNpcmVkQ3B1c2V0IgogICAgZmkKCiAgICAjIERldGVjdCBzdGVhZHk
tc3RhdGUgcG9kIGNvdW50CiAgICBjY291bnQ9JChjcmljdGwgcHMgfCB3YyAtbCkKICAgIGlmIHN0ZW
FkeXN0YXRlICRsYXN0Q2NvdW50ICRjY291bnQ7IHRoZW4KICAgICAgKChzdGVhZHlTdGF0ZVRpb
WUgKz0gcykpCiAgICAgIGVjaG8gIlN0ZWFkeS1zdGF0ZSBmb3IgJHtzdGVhZHlTdGF0ZVRpbWV9cy8
ke1NURUFEWV9TVEFURV9XSU5ET1d9cyIKICAgICAgaWYgW1sgJHN0ZWFkeVN0YXRlVGltZSAtZ2
UgJFNURUFEWV9TVEFURV9XSU5ET1cgXV07IHRoZW4KICAgICAgICBsb2dnZXIgIlJlY292ZXJ5OiB
TdGVhZHktc3RhdGUgKCsvLSAkU1RFQURZX1NUQVRFX1RIUkVTSE9MRCkgZm9yICR7U1RFQU
RZX1NUQVRFX1dJTkRPV31zOiBEb25lIgogICAgICAgIHJldHVybiAwCiAgICAgIGZpCiAgICBlbHNlCiAg
ICAgIGlmIFtbICRzdGVhZHlTdGF0ZVRpbWUgLWd0IDAgXV07IHRoZW4KICAgICAgICBlY2hvICJSZX
NldHRpbmcgc3RlYWR5LXN0YXRlIHRpbWVyIgogICAgICAgIHN0ZWFkeVN0YXRlVGltZT0wCiAgICAgI
GZpCiAgICBmaQogICAgbGFzdENjb3VudD0kY2NvdW50CiAgZG9uZQogIGxvZ2dlciAiUmVjb3Zlcnk6I
FJlY292ZXJ5IENvbXBsZXRlIFRpbWVvdXQiCn0KCm1haW4oKSB7CiAgaWYgISB1bnJlc3RyaWN0ZW
RDcHVzZXQgPiYvZGV2L251bGw7IHRoZW4KICAgIGxvZ2dlciAiUmVjb3Zlcnk6IE5vIHVucmVzdHJpY3
RlZCBDcHVzZXQgY291bGQgYmUgZGV0ZWN0ZWQiCiAgICByZXR1cm4gMQogIGZpCgogIGlmICE
gcmVzdHJpY3RlZENwdXNldCA+Ji9kZXYvbnVsbDsgdGhlbgogICAgbG9nZ2VyICJSZWNvdmVyeTogT
m8gcmVzdHJpY3RlZCBDcHVzZXQgaGFzIGJlZW4gY29uZmlndXJlZC4gIFdlIGFyZSBhbHJlYWR5IHJ1
bm5pbmcgdW5yZXN0cmljdGVkLiIKICAgIHJldHVybiAwCiAgZmkKCiAgIyBFbnN1cmUgd2UgcmVzZXQg
dGhlIENQVSBhZmZpbml0eSB3aGVuIHdlIGV4aXQgdGhpcyBzY3JpcHQgZm9yIGFueSByZWFzb24KI
CAjIFRoaXMgd2F5IGVpdGhlciBhZnRlciB0aGUgdGltZXIgZXhwaXJlcyBvciBhZnRlciB0aGUgcHJvY2Vzc
yBpcyBpbnRlcnJ1cHRlZAogICMgdmlhIF5DIG9yIFNJR1RFUk0sIHdlIHJldHVybiB0aGluZ3MgYmFjayB0
byB0aGUgd2F5IHRoZXkgc2hvdWxkIGJlLgogIHRyYXAgc2V0UmVzdHJpY3RlZCBFWElUCgogIGxvZ2
dlciAiUmVjb3Zlcnk6IFJlY292ZXJ5IE1vZGUgU3RhcnRpbmciCiAgc2V0VW5yZXN0cmljdGVkCiAgd2Fpd
EZvclJlYWR5Cn0KCmlmIFtbICIke0JBU0hfU09VUkNFWzBdfSIgPSAiJHswfSIgXV07IHRoZW4KICBtYW
luICIke0B9IgogIGV4aXQgJD8KZmkK
        mode: 493
        path: /usr/local/bin/accelerated-container-startup.sh
  systemd:
    units:
    - contents: |
        [Unit]
        Description=Unlocks more CPUs for critical system processes during container startup

        [Service]
        Type=simple
        ExecStart=/usr/local/bin/accelerated-container-startup.sh
        # Maximum wait time is 600s = 10m:
        Environment=MAXIMUM_WAIT_TIME=600

        # Steady-state threshold = 2%
        # Allowed values:
        #  4  - absolute pod count (+/-)
        #  4% - percent change (+/-)
        #  -1 - disable the steady-state check
        # Note: '%' must be escaped as '%%' in systemd unit files
        Environment=STEADY_STATE_THRESHOLD=2%%

        # Steady-state window = 120s
        # If the running pod count stays within the given threshold for this time
        # period, return CPU utilization to normal before the maximum wait time has
        # expires
        Environment=STEADY_STATE_WINDOW=120

        # Steady-state minimum = 40
        # Increasing this will skip any steady-state checks until the count rises above
        # this number to avoid false positives if there are some periods where the
        # count doesn't increase but we know we can't be at steady-state yet.
        Environment=STEADY_STATE_MINIMUM=40

        [Install]
        WantedBy=multi-user.target
      enabled: true
      name: accelerated-container-startup.service
    - contents: |
        [Unit]
        Description=Unlocks more CPUs for critical system processes during container shutdown
        DefaultDependencies=no

        [Service]
        Type=simple
        ExecStart=/usr/local/bin/accelerated-container-startup.sh

        # Maximum wait time is 600s = 10m:
        Environment=MAXIMUM_WAIT_TIME=600

        # Steady-state threshold
        # Allowed values:
        #  4  - absolute pod count (+/-)
        #  4% - percent change (+/-)
        #  -1 - disable the steady-state check
        # Note: '%' must be escaped as '%%' in systemd unit files
        Environment=STEADY_STATE_THRESHOLD=-1

        # Steady-state window = 60s
        # If the running pod count stays within the given threshold for this time
        # period, return CPU utilization to normal before the maximum wait time has
        # expires
        Environment=STEADY_STATE_WINDOW=60

        [Install]
        WantedBy=shutdown.target reboot.target halt.target
      enabled: true
      name: accelerated-container-shutdown.service
```

17.6.6.5. Automatic kernel crash dumps with kdump

The **kdump** Linux kernel feature creates a kernel crash dump when the kernel crashes. The **kdump** feature is enabled with the following **MachineConfig** CRs.

Recommended MachineConfig CR to remove ice driver from control plane kdump logs (05-kdump-config-master.yaml)

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: 05-kdump-config-master
spec:
  config:
    ignition:
      version: 3.2.0
    systemd:
      units:
      - enabled: true
        name: kdump-remove-ice-module.service
        contents: |
          [Unit]
          Description=Remove ice module when doing kdump
          Before=kdump.service
          [Service]
          Type=oneshot
          RemainAfterExit=true
          ExecStart=/usr/local/bin/kdump-remove-ice-module.sh
          [Install]
          WantedBy=multi-user.target
    storage:
      files:
      - contents:
          source: data:text/plain;charset=utf-
8;base64,IyEvdXNyL2Jpbi9lbnYgYmFzaAoKIyBUaGlzIHNjcmlwdCByZW1vdmVzIHRoZSBpY2UgbW9k
dWxlIGZyb20ga2R1bXAgdG8gcHJldmVudCBrZHVtcCBmYWlsdXJlcyBvbiBjZXJ0YWluIHNlcnZlcnMuCi
MgVGhpcyBpcyBhIHRlbXBvcmFyeSB3b3JrYXJvdW5kIGZvciBSSEVMUExBTi0xMzgyMzYgYW5kIGNh
iBiZSByZW1vdmVkIHdoZW4gdGhhdCBpc3N1ZSBpcwojIGZpeGVkLgoKc2V0IC14CgpTRUQ9Ii91c3Iv
YmluL3NlZCIKR1JFUD0iL3Vzci9iaW4vZ3JlcCIKCiMgb3ZlcnJpZGUgZm9yIHRlc3RpbmcgcHVycG9zZX
MKS0RVTVBfQ09ORj0iJHsxOi0vZXRjL3N5c2NvbmZpZy9rZHVtcH0iClJFTU9WRV9JQ0VfU1RSPSJtb
2R1bGVfYmxhY2tsaXN0PWljZSIKCiMgZXhpdCBpZiBmaWxlIGRvZXNuJ3QgZXhpc3QKWyAhIC1mIC
R7S0RVTVBfQ09ORn0gXSAmJiBleGl0IDAKCiMgZXhpdCBpZiBmaWxlIGFscmVhZHkgdXBkYXRlZAok
e0dSRVB9IC1GcSAke1JFTU9WRV9JQ0VfU1RSfSAke0tEVU1QX0NPTkZ9ICYmIGV4aXQgMAoKIyB
UYXJnZXQgbGluZSBsb29rcyBzb21ldGhpbmcgbGlrZSB0aGlzOgojIEtEVU1QX0NPTU1BTkRMSU5FX
0FQUEVORD0iaXJxcG9sbCBucl9jcHVzPTEgLi4uIGhlc3RfZGlzYWJsZSIKIyBVc2Ugc2VkIHRvIG1hdG
NoIGV2ZXJ5dGhpbmcgYmV0d2VlbiB0aGUgcXVvdGVzIGFuZCBhcHBlbmQgdGhlIFJFTU9WRV9JQ0
VfU1RSIHRvIGl0CiR7U0VEfSAtaSAncy9eS0RVTVBfQ09NTUFOREExJTkVfQVBQRU5EPSJbXiJdKi8m
```

```
ICcke1JFTU9WRV9JQ0VfU1RSfScvJyAke0tEVU1QX0NPTkZ9IHx8IGV4aXQgMAo=
          mode: 448
          path: /usr/local/bin/kdump-remove-ice-module.sh
```

Recommended control plane node kdump configuration (**06-kdump-master.yaml**)

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: 06-kdump-enable-master
spec:
  config:
    ignition:
      version: 3.2.0
    systemd:
      units:
        - enabled: true
          name: kdump.service
  kernelArguments:
    - crashkernel=512M
```

Recommended **MachineConfig** CR to remove ice driver from worker node kdump logs (**05-kdump-config-worker.yaml**)

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: 05-kdump-config-worker
spec:
  config:
    ignition:
      version: 3.2.0
    systemd:
      units:
        - enabled: true
          name: kdump-remove-ice-module.service
          contents: |
            [Unit]
            Description=Remove ice module when doing kdump
            Before=kdump.service
            [Service]
            Type=oneshot
            RemainAfterExit=true
            ExecStart=/usr/local/bin/kdump-remove-ice-module.sh
            [Install]
            WantedBy=multi-user.target
    storage:
      files:
        - contents:
            source: data:text/plain;charset=utf-
8;base64,IyEvdXNyL2Jpbi9lbnYgYmFzaAoKIyBUaGlzIHNjcmlwdCByZW1vdmVzIHRoZSBpY2UgbW9k
```

```
dWxlIGZyb20ga2R1bXAgdG8gcHJldmVudCBrZHVtcCBmYWlsdXJlcyBvbiBjZXJ0YWluIHNlcnZlcnMuCi
MgVGhpcyBpcyBhIHRlbXBvcmFyeSB3b3JrYXJvdW5kIGZvciBSSEVMUExBTi0xMzgyMzYgYW5kIGNh
iBiZSByZW1vdmVkIHdoZW4gdGhhdCBpc3N1ZSBpcwojIGZpeGVkLgoKc2V0IC14CgpTRUQ9Ii91c3Iv
YmluL3NlZCIKR1JFUD0iL3Vzci9iaW4vZ3JlcCIKCiMgb3ZlcnJpZGUgZm9yIHRlc3RpbmcgcHVycG9zZX
MKS0RVTVBfQ09ORj0iJHsxOi0vZXRjL3N5c2NvbmZpZy9rZHVtcH0iClJFTU9WRV9JQ0VfU1RSPSJtb
2R1bGVfYmxhY2tsaXN0PWljZSIKCiMgZXhpdCBpZiBmaWxlIGRvZXNuJ3QgZXhpc3QKWyAhIC1mIC
R7S0RVTVBfQ09ORn0gXSAmJiBleGl0IDAKCiMgZXhpdCBpZiBmaWxlIGFscmVhZHkgdXBkYXRlZAok
e0dSRVB9IC1GcSAke1JFTU9WRV9JQ0VfU1RSfSAke0tEVU1QX0NPTkZ9ICYmIGV4aXQgMAoKIyB
UYXJnZXQgbGluZSBsb29rcyBzb21ldGhpbmcgbGlrZSB0aGlzOgojIEtEVU1QX0NPTU1BTkRMSU5FX
0FQUEVORD0iaXJxcG9sbCBucl9jcHVzPTEgLi4uIGhlc3RfZGlzYWJsZSIKIyBVc2Ugc2VkIHRvIG1hdG
NoIGV2ZXJ5dGhpbmcgYmV0d2VlbiB0aGUgcXVvdGVzIGFuZCBhcHBlbmQgdGhlIFJFTU9WRV9JQ0
VfU1RSIHRvIGl0CiR7U0VEfSAtaSAncy9eS0RVTVBfQ09NTUFORExJTkVfQVBQRU5EPSJbXiJdKi8m
ICcke1JFTU9WRV9JQ0VfU1RSfScvJyAke0tEVU1QX0NPTkZ9IHx8IGV4aXQgMAo=
        mode: 448
        path: /usr/local/bin/kdump-remove-ice-module.sh
```

Recommended kdump worker node configuration (06-kdump-worker.yaml)

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: 06-kdump-enable-worker
spec:
  config:
    ignition:
      version: 3.2.0
    systemd:
      units:
        - enabled: true
          name: kdump.service
  kernelArguments:
    - crashkernel=512M
```

17.6.7. Recommended postinstallation cluster configurations

When the cluster installation is complete, the ZTP pipeline applies the following custom resources (CRs) that are required to run DU workloads.

NOTE

In {ztp} v4.10 and earlier, you configure UEFI secure boot with a **MachineConfig** CR. This is no longer required in {ztp} v4.11 and later. In v4.11, you configure UEFI secure boot for single-node OpenShift clusters by updating the **spec.clusters.nodes.bootMode** field in the **SiteConfig** CR that you use to install the cluster. For more information, see Deploying a managed cluster with SiteConfig and {ztp} .

17.6.7.1. Operator namespaces and Operator groups

Single-node OpenShift clusters that run DU workloads require the following **OperatorGroup** and **Namespace** custom resources (CRs):

- Local Storage Operator

- Logging Operator
- PTP Operator
- SR-IOV Network Operator

The following YAML summarizes these CRs:

Recommended Operator Namespace and OperatorGroup configuration

```
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
---
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
---
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
---
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
```

17.6.7.2. Operator subscriptions

Single-node OpenShift clusters that run DU workloads require the following **Subscription** CRs. The subscription provides the location to download the following Operators:

- Local Storage Operator
- Logging Operator
- PTP Operator
- SR-IOV Network Operator

Recommended Operator subscriptions

```
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"
  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"
  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"
  name: sriov-network-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  installPlanApproval: Manual
```

1 Specify the channel to get the Operator from. **stable** is the recommended channel.

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.

17.6.7.3. Cluster logging and log forwarding

Single-node OpenShift clusters that run DU workloads require logging and log forwarding for debugging. The following example YAML illustrates the required **ClusterLogging** and **ClusterLogForwarder** CRs.

Recommended cluster logging and log forwarding configuration

```
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: {}
      name: infra-logs
  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 **ClusterLogging** instance or creates the instance if it does not exist.

2 Updates the existing **ClusterLogForwarder** instance or creates the instance if it does not exist.

3 Specifies the URL of the Kafka server where the logs are forwarded to.

17.6.7.4. Performance profile

Single-node OpenShift clusters that run DU workloads require a Node Tuning Operator performance profile to use real-time host capabilities and services.

NOTE

In earlier versions of OpenShift Container Platform, the Performance Addon Operator was used to implement automatic tuning to achieve low latency performance for OpenShift applications. In OpenShift Container Platform 4.11 and later, this functionality is part of the Node Tuning Operator.

The following example **PerformanceProfile** CR illustrates the required cluster configuration.

Recommended performance profile configuration

```
apiVersion: performance.openshift.io/v2
```

```
kind: PerformanceProfile
metadata:
  name: openshift-node-performance-profile 1
spec:
  additionalKernelArgs:
  - "rcupdate.rcu_normal_after_boot=0"
  - "efi=runtime" 2
  cpu:
    isolated: 2-51,54-103 3
    reserved: 0-1,52-53  4
  hugepages:
    defaultHugepagesSize: 1G
    pages:
      - count: 32 5
        size: 1G  6
        node: 0 7
  machineConfigPoolSelector:
    pools.operator.machineconfiguration.openshift.io/master: ""
  nodeSelector:
    node-role.kubernetes.io/master: ""
  numa:
    topologyPolicy: "restricted"
  realTimeKernel:
    enabled: true   8
```

1 Ensure that the value for **name** matches that specified in the **spec.profile.data** field of **TunedPerformancePatch.yaml** and the **status.configuration.source.name** field of **validatorCRs/informDuValidator.yaml**.

2 Configures UEFI secure boot for the cluster host.

3 Set the isolated CPUs. Ensure all of the Hyper-Threading pairs match.

> **IMPORTANT**
>
> The reserved and isolated CPU pools must not overlap and together must span all available cores. CPU cores that are not accounted for cause an undefined behaviour in the system.

4 Set the reserved CPUs. When workload partitioning is enabled, system processes, kernel threads, and system container threads are restricted to these CPUs. All CPUs that are not isolated should be reserved.

5 Set the number of huge pages.

6 Set the huge page size.

7 Set **node** to the NUMA node where the **hugepages** are allocated.

8 Set **enabled** to **true** to install the real-time Linux kernel.

17.6.7.5. PTP

Single-node OpenShift clusters use Precision Time Protocol (PTP) for network time synchronization. The following example **PtpConfig** CR illustrates the required PTP slave configuration.

Recommended PTP configuration

```
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 1
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 2.0
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
      ptp4lOpts: -2 -s --summary_interval -4
recommend:
  - match:
      - nodeLabel: node-role.kubernetes.io/master
    priority: 4
    profile: slave
```

1 Sets the interface used to receive the PTP clock signal.

17.6.7.6. Extended Tuned profile

Single-node OpenShift clusters that run DU workloads require additional performance tuning configurations necessary for high-performance workloads. The following example **Tuned** CR extends the **Tuned** profile:

Recommended extended Tuned profile configuration

```
apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
  name: performance-patch
  namespace: openshift-cluster-node-tuning-operator
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=2-51,54-103
        [sysctl]
        kernel.timer_migration=1
        [scheduler]
        group.ice-ptp=0:f:10:*:ice-ptp.*
        [service]
        service.stalld=start,enable
        service.chronyd=stop,disable
      name: performance-patch
  recommend:
    - machineConfigLabels:
        machineconfiguration.openshift.io/role: master
      priority: 19
      profile: performance-patch
```

17.6.7.7. SR-IOV

Single root I/O virtualization (SR-IOV) is commonly used to enable the fronthaul and the midhaul networks. The following YAML example configures SR-IOV for a single-node OpenShift cluster.

Recommended SR-IOV configuration

```
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 ②
  isRdma: false
  nicSelector:
    pfNames:
      - ens7f0 ③
  nodeSelector:
    node-role.kubernetes.io/master: ""
  numVfs: 8 ④
  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 ⑤
---
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: sriov-nnp-du-fh
  namespace: openshift-sriov-network-operator
spec:
  deviceType: netdevice 6
  isRdma: true
  nicSelector:
    pfNames:
      - ens5f0 7
  nodeSelector:
    node-role.kubernetes.io/master: ""
  numVfs: 8 8
  priority: 10
  resourceName: du_fh
```

1. Specifies the VLAN for the midhaul network.
2. Select either **vfio-pci** or **netdevice**, as needed.
3. Specifies the interface connected to the midhaul network.
4. Specifies the number of VFs for the midhaul network.
5. The VLAN for the fronthaul network.
6. Select either **vfio-pci** or **netdevice**, as needed.
7. Specifies the interface connected to the fronthaul network.
8. Specifies the number of VFs for the fronthaul network.

17.6.7.8. 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. The following **Console** custom resource (CR) example disables the console.

Recommended console configuration

```
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
```

17.6.7.9. Alertmanager

Single-node OpenShift clusters that run DU workloads require reduced CPU resources consumed by the OpenShift Container Platform monitoring components. The following **ConfigMap** custom resource (CR) disables Alertmanager.

Recommended cluster monitoring configuration

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    alertmanagerMain:
      enabled: false
    prometheusK8s:
       retention: 24h
```

17.6.7.10. Operator Lifecycle Manager

Single-node OpenShift clusters that run distributed unit workloads require consistent access to CPU resources. Operator Lifecycle Manager (OLM) collects performance data from Operators at regular intervals, resulting in an increase in CPU utilisation. The following **ConfigMap** custom resource (CR) disables the collection of Operator performance data by OLM.

Recommended cluster OLM configuration (ReduceOLMFootprint.yaml)

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: collect-profiles-config
  namespace: openshift-operator-lifecycle-manager
data:
  pprof-config.yaml: |
    disabled: True
```

17.6.7.11. Network diagnostics

Single-node OpenShift clusters that run DU workloads require less inter-pod network connectivity checks to reduce the additional load created by these pods. The following custom resource (CR) disables these checks.

Recommended network diagnostics configuration

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  disableNetworkDiagnostics: true
```

Additional resources

- Deploying far edge sites using ZTP

17.7. VALIDATING SINGLE-NODE OPENSHIFT CLUSTER TUNING FOR VDU APPLICATION WORKLOADS

Before you can deploy virtual distributed unit (vDU) applications, you need to tune and configure the cluster host firmware and various other cluster configuration settings. Use the following information to validate the cluster configuration to support vDU workloads.

Additional resources

- For more information about single-node OpenShift clusters tuned for vDU application deployments, see Reference configuration for deploying vDUs on single-node OpenShift.

17.7.1. Recommended firmware configuration for vDU cluster hosts

Use the following table as the basis to configure the cluster host firmware for vDU applications running on OpenShift Container Platform 4.12.

NOTE

The following table is a general recommendation for vDU cluster host firmware configuration. Exact firmware settings will depend on your requirements and specific hardware platform. Automatic setting of firmware is not handled by the zero touch provisioning pipeline.

Table 17.8. Recommended cluster host firmware settings

Firmware 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.
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.

Firmware setting	Configuration	Description
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	Enabled or Disabled	Enable OS-controlled P-States to allow power saving configurations. 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 firmware for the host. These settings are relevant to bare-metal environments.

NOTE

Enable both **C-states** and OS-controlled **P-States** to allow per pod power management.

17.7.2. 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 4.12 clusters.

17.7.2.1. Recommended cluster MachineConfig CRs

Check that the **MachineConfig** custom resources (CRs) that you extract from the **ztp-site-generate** container are applied in the cluster. The CRs can be found in the extracted **out/source-crs/extra-manifest/** folder.

The following **MachineConfig** CRs from the **ztp-site-generate** container configure the cluster host:

Table 17.9. 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.
03-sctp-machine-config-master.yaml, **03-sctp-machine-config-worker.yaml**	Loads the SCTP kernel module. These **MachineConfig** CRs are optional and can be omitted if you do not require this kernel module.
01-container-mount-ns-and-kubelet-conf-master.yaml, **01-container-mount-ns-and-kubelet-conf-worker.yaml**	Configures the container mount namespace and Kubelet configuration.
04-accelerated-container-startup-master.yaml, **04-accelerated-container-startup-worker.yaml**	Configures accelerated startup for the cluster.
06-kdump-master.yaml, **06-kdump-worker.yaml**	Configures **kdump** for the cluster.

Additional resources

- Extracting source CRs from the ztp-site-generate container

17.7.2.2. Recommended cluster Operators

The following Operators are required for clusters running virtualized distributed unit (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

17.7.2.3. Recommended cluster kernel configuration

Always use the latest supported real-time kernel version in your cluster. Ensure that you apply the following configurations in the cluster:

1. Ensure that the following **additionalKernelArgs** are set in the cluster performance profile:

```
spec:
  additionalKernelArgs:
  - "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.chronyd=stop,disable
```

1 Listed CPUs depend on the host hardware configuration, specifically the number of available CPUs in the system and the CPU topology.

17.7.2.4. 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
```

17.7.3. 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 4.12 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 OperatorHub 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 1
redhat-marketplace -- {"effect": "PreferredDuringScheduling"}
redhat-operators -- {"effect": "PreferredDuringScheduling"}
```

1 **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 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.fffe.7a7020-1 seq 0 RESPONSE MANAGEMENT PORT_DATA_SET
    portIdentity            3cecef.fffe.7a7020-1
    portState               SLAVE
    logMinDelayReqInterval  -4
    peerMeanPathDelay       0
    logAnnounceInterval     1
    announceReceiptTimeout  3
    logSyncInterval         0
    delayMechanism          1
    logMinPdelayReqInterval 0
    versionNumber           2
  3cecef.fffe.7a7020-2 seq 0 RESPONSE MANAGEMENT PORT_DATA_SET
    portIdentity            3cecef.fffe.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.fffe.7a7020-0 seq 0 RESPONSE MANAGEMENT TIME_STATUS_NP
    master_offset              10 ❶
    ingress_time               1657275432697400530
    cumulativeScaledRateOffset +0.000000000
    scaledLastGmPhaseChange    0
    gmTimeBaseIndicator        0
    lastGmPhaseChange          0x0000'0000000000000000.0000
    gmPresent                  true ❷
    gmIdentity                 3c2c30.ffff.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}{'\n'}"
```

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}{'\n'}"
```

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 tuneds.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.chronyd=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
    OPENSHIFT_MAX_HOUSEKEEPING_INTERVAL_DURATION
    ```

    Example output

    ```
    Environment="OPENSHIFT_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

```
0-3
```

NOTE

Depending on your workload requirements, you might require additional reserved CPUs to be allocated.

17.8. ADVANCED MANAGED CLUSTER CONFIGURATION WITH SITECONFIG RESOURCES

You can use **SiteConfig** custom resources (CRs) to deploy custom functionality and configurations in your managed clusters at installation time.

17.8.1. 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.

17.8.2. 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-4.12"
  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>" 1
  extraManifests:
    filter:
      inclusionDefault: exclude 2
      include:
        - custom-sctp-machine-config-worker.yaml
```

1 Replace **<custom_manifest_folder>** with the name of the folder that contains the custom installation CRs, for example, **user-custom-manifest/**.

2 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
```

17.9. ADVANCED MANAGED CLUSTER CONFIGURATION WITH POLICYGENTEMPLATE RESOURCES

You can use **PolicyGenTemplate** CRs to deploy custom functionality in your managed clusters.

17.9.1. Deploying additional changes to clusters

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 custom resources (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 make 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

- Customizing extra installation manifests in the ZTP GitOps pipeline

17.9.2. Using PolicyGenTemplate CRs to override source CRs content

PolicyGenTemplate custom resources (CRs) allow you to overlay additional configuration details on top of the base source CRs provided with the GitOps plugin 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:v4.12.1 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 RHACM 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.

17.9.3. Adding custom content to the GitOps ZTP pipeline

Perform the following procedure to add new content to the ZTP pipeline.

Procedure

1. Create a subdirectory named **source-crs** in the directory that contains the **kustomization.yaml** file for the **PolicyGenTemplate** custom resource (CR).

2. Add your custom CRs to the **source-crs** subdirectory, as shown in the following example:

```
example
└── policygentemplates
    ├── dev.yaml
    ├── kustomization.yaml
    ├── mec-edge-sno1.yaml
    ├── sno.yaml
    └── source-crs (1)
        ├── PaoCatalogSource.yaml
        ├── PaoSubscription.yaml
        ├── custom-crs
        |   ├── apiserver-config.yaml
        |   └── disable-nic-lldp.yaml
        └── elasticsearch
            ├── ElasticsearchNS.yaml
            └── ElasticsearchOperatorGroup.yaml
```

(1) The **source-crs** subdirectory must be in the same directory as the **kustomization.yaml** file.

IMPORTANT

To use your own resources, ensure that the custom CR names differ from the default source CRs provided in the ZTP container.

3. Update the required **PolicyGenTemplate** CRs to include references to the content you added in the **source-crs/custom-crs** and **source-crs/elasticsearch** directories. For example:

```
apiVersion: ran.openshift.io/v1
kind: PolicyGenTemplate
metadata:
  name: "group-dev"
  namespace: "ztp-clusters"
spec:
  bindingRules:
    dev: "true"
  mcp: "master"
  sourceFiles:
    # These policies/CRs come from the internal container Image
    #Cluster Logging
    - fileName: ClusterLogNS.yaml
      remediationAction: inform
      policyName: "group-dev-cluster-log-ns"
    - fileName: ClusterLogOperGroup.yaml
      remediationAction: inform
      policyName: "group-dev-cluster-log-operator-group"
    - fileName: ClusterLogSubscription.yaml
      remediationAction: inform
      policyName: "group-dev-cluster-log-sub"
    #Local Storage Operator
    - fileName: StorageNS.yaml
      remediationAction: inform
      policyName: "group-dev-lso-ns"
    - fileName: StorageOperGroup.yaml
      remediationAction: inform
      policyName: "group-dev-lso-operator-group"
    - fileName: StorageSubscription.yaml
      remediationAction: inform
      policyName: "group-dev-lso-sub"
    #These are custom local polices that come from the source-crs directory in the git repo
    # Performance Addon Operator
    - fileName: PaoSubscriptionNS.yaml
      remediationAction: inform
      policyName: "group-dev-pao-ns"
    - fileName: PaoSubscriptionCatalogSource.yaml
      remediationAction: inform
      policyName: "group-dev-pao-cat-source"
      spec:
        image: <image_URL_here>
    - fileName: PaoSubscription.yaml
      remediationAction: inform
      policyName: "group-dev-pao-sub"
    #Elasticsearch Operator
    - fileName: elasticsearch/ElasticsearchNS.yaml ❶
      remediationAction: inform
      policyName: "group-dev-elasticsearch-ns"
    - fileName: elasticsearch/ElasticsearchOperatorGroup.yaml
      remediationAction: inform
      policyName: "group-dev-elasticsearch-operator-group"
    #Custom Resources
    - fileName: custom-crs/apiserver-config.yaml ❷
    remediationAction: inform
    policyName: "group-dev-apiserver-config"
  - fileName: custom-crs/disable-nic-lldp.yaml
    remediationAction: inform
    policyName: "group-dev-disable-nic-lldp"
```

1 2 Set the **fileName** field to include the relative path to the file from the **/source-crs** parent directory.

4. Commit the **PolicyGenTemplate** change in Git, and then push to the Git repository that is monitored by the GitOps ZTP Argo CD policies application.

5. Update the **ClusterGroupUpgrade** CR to include the changed **PolicyGenTemplate** and save it as **cgu-test.yaml**. The following example shows a generated **cgu-test.yaml** file.

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: custom-source-cr
  namespace: ztp-clusters
spec:
  managedPolicies:
    - group-dev-config-policy
  enable: true
  clusters:
  - cluster1
  remediationStrategy:
    maxConcurrency: 2
    timeout: 240
```

6. Apply the updated **ClusterGroupUpgrade** CR by running the following command:

```
$ oc apply -f cgu-test.yaml
```

Verification

- Check that the updates have succeeded by running the following command:

```
$ oc get cgu -A
```

Example output

```
NAMESPACE     NAME               AGE   STATE        DETAILS
ztp-clusters  custom-source-cr   6s    InProgress   Remediating non-compliant policies
ztp-install   cluster1           19h   Completed    All clusters are compliant with all the managed
policies
```

17.9.4. 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"}
```

17.9.5. Signalling ZTP cluster deployment completion with validator inform policies

Create a validator inform policy that signals 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 OpenShift clusters, three-node clusters, and standard clusters.

Procedure

1. Create a standalone **PolicyGenTemplate** custom resource (CR) that contains the source file **validatorCRs/informDuValidator.yaml**. You only need one standalone **PolicyGenTemplate** CR for each cluster type. For example, this CR applies a validator inform policy for single-node OpenShift clusters:

Example single-node cluster validator inform policy CR (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
```

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. This value should match the **namespace** used in the group **PolicyGenTemplates**.

3. The **group-du-*** label defined in **bindingRules** must exist in the **SiteConfig** files.

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. **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. Optional. The default value is **inform**.

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 **group-du-sno-validator-du-policy**.

2. Commit the **PolicyGenTemplate** CR file in your Git repository and push the changes.

Additional resources

- Upgrading GitOps ZTP
- Preparing the GitOps ZTP site configuration repository

17.9.6. Configuring PTP events with PolicyGenTemplate CRs

You can use the GitOps ZTP pipeline to configure PTP events that use HTTP or AMQP transport.

> **NOTE**
>
> HTTP transport is the default transport for PTP and bare-metal events. Use HTTP transport instead of AMQP for PTP and bare-metal events where possible. AMQ Interconnect is EOL from 30 June 2024. Extended life cycle support (ELS) for AMQ Interconnect ends 29 November 2029. For more information see, Red Hat AMQ Interconnect support status.

17.9.6.1. Configuring PTP events that use HTTP transport

You can configure PTP events that use HTTP transport on managed clusters that you deploy with the GitOps Zero Touch Provisioning (ZTP) pipeline.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have logged in as a user with **cluster-admin** privileges.
- You have created a Git repository where you manage your custom site configuration data.

Procedure

1. 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 transport host:

```
- fileName: PtpOperatorConfigForEvent.yaml
  policyName: "config-policy"
  spec:
    daemonNodeSelector: {}
    ptpEventConfig:
      enableEventPublisher: true
      transportHost: http://ptp-event-publisher-service-NODE_NAME.openshift-
ptp.svc.cluster.local:9043
```

   NOTE

   In OpenShift Container Platform 4.12 or later, you do not need to set the **transportHost** field in the **PtpOperatorConfig** resource when you use HTTP transport with PTP events.

   b. Configure the **linuxptp** and **phc2sys** for the PTP clock type and interface. For example, add the following stanza into **.sourceFiles**:

```
- fileName: PtpConfigSlave.yaml 1
  policyName: "config-policy"
  metadata:
    name: "du-ptp-slave"
  spec:
    profile:
    - name: "slave"
      interface: "ens5f1" 2
      ptp4lOpts: "-2 -s --summary_interval -4" 3
      phc2sysOpts: "-a -r -m -n 24 -N 8 -R 16" 4
    ptpClockThreshold: 5
      holdOverTimeout: 30 #secs
      maxOffsetThreshold: 100  #nano secs
      minOffsetThreshold: -100 #nano secs
```

1 Can be one of **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.

2. Merge any other required changes and files with your custom site repository.

3. Push the changes to your site configuration repository to deploy PTP fast events to new sites using GitOps ZTP.

Additional resources

- Using PolicyGenTemplate CRs to override source CRs content

17.9.6.2. Configuring PTP events that use AMQP transport

You can configure PTP events that use AMQP transport on managed clusters that you deploy with the GitOps Zero Touch Provisioning (ZTP) pipeline.

> **NOTE**
>
> HTTP transport is the default transport for PTP and bare-metal events. Use HTTP transport instead of AMQP for PTP and bare-metal events where possible. AMQ Interconnect is EOL from 30 June 2024. Extended life cycle support (ELS) for AMQ Interconnect ends 29 November 2029. For more information see, Red Hat AMQ Interconnect support status.

Prerequisites

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

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

- You have created 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"
spec:
  daemonNodeSelector: {}
  ptpEventConfig:
    enableEventPublisher: true
    transportHost: "amqp://amq-router.amq-router.svc.cluster.local"
```

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**.

❷ Device specific interface name.

❸ You must append the **--summary_interval -4** value to **ptp4lOpts** in **.spec.sourceFiles.spec.profile** to enable PTP fast events.

❹ Required **phc2sysOpts** values. **-m** prints messages to **stdout**. The **linuxptp-daemon** **DaemonSet** parses the logs and generates Prometheus metrics.

❺ 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.

Additional resources

- Installing the AMQ messaging bus

17.9.7. Configuring bare-metal events with PolicyGenTemplate CRs

You can use the GitOps ZTP pipeline to configure bare-metal events that use HTTP or AMQP transport.

> **NOTE**
>
> HTTP transport is the default transport for PTP and bare-metal events. Use HTTP transport instead of AMQP for PTP and bare-metal events where possible. AMQ Interconnect is EOL from 30 June 2024. Extended life cycle support (ELS) for AMQ Interconnect ends 29 November 2029. For more information see, Red Hat AMQ Interconnect support status.

17.9.7.1. Configuring bare-metal events that use HTTP transport

You can configure bare-metal events that use HTTP transport on managed clusters that you deploy with the GitOps Zero Touch Provisioning (ZTP) pipeline.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have logged in as a user with **cluster-admin** privileges.
- You have created a Git repository where you manage your custom site configuration data.

Procedure

1. Configure the Bare Metal Event Relay Operator by adding the following YAML to **spec.sourceFiles** in the **common-ranGen.yaml** file:

```
# 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 **HardwareEvent** CR to **spec.sourceFiles** in your specific group configuration file, for example, in the **group-du-sno-ranGen.yaml** file:

```
- fileName: HardwareEvent.yaml 1
  policyName: "config-policy"
  spec:
    nodeSelector: {}
    transportHost: "http://hw-event-publisher-service.openshift-bare-metal-events.svc.cluster.local:9043"
    logLevel: "info"
```

1 Each baseboard management controller (BMC) requires a single **HardwareEvent** CR only.

> **NOTE**
>
> In OpenShift Container Platform 4.12 or later, you do not need to set the **transportHost** field in the **HardwareEvent** custom resource (CR) when you use HTTP transport with bare-metal events.

3. Merge any other required changes and files with your custom site repository.

4. Push the changes to your site configuration repository to deploy bare-metal events to new sites with 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

- Installing the Bare Metal Event Relay using the CLI
- Creating the bare-metal event and Secret CRs

17.9.7.2. Configuring bare-metal events that use AMQP transport

You can configure bare-metal events that use AMQP transport on managed clusters that you deploy with the GitOps Zero Touch Provisioning (ZTP) pipeline.

> **NOTE**
>
> HTTP transport is the default transport for PTP and bare-metal events. Use HTTP transport instead of AMQP for PTP and bare-metal events where possible. AMQ Interconnect is EOL from 30 June 2024. Extended life cycle support (ELS) for AMQ Interconnect ends 29 November 2029. For more information see, Red Hat AMQ Interconnect support status.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have logged in 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 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 Rely 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**.

NOTE

Each baseboard management controller (BMC) requires a single **HardwareEvent** resource only.

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>"
```

17.9.8. Configuring the Image Registry Operator for local caching of images

OpenShift Container Platform manages image caching using a local registry. In edge computing use cases, clusters are often subject to bandwidth restrictions when communicating with centralized image registries, which might result in long image download times.

Long download times are unavoidable during initial deployment. Over time, there is a risk that CRI-O will erase the **/var/lib/containers/storage** directory in the case of an unexpected shutdown. To address long image download times, you can create a local image registry on remote managed clusters using GitOps ZTP. This is useful in Edge computing scenarios where clusters are deployed at the far edge of the network.

Before you can set up the local image registry with GitOps ZTP, you need to configure disk partitioning in the **SiteConfig** CR that you use to install the remote managed cluster. After installation, you configure the local image registry using a **PolicyGenTemplate** CR. Then, the ZTP pipeline creates Persistent Volume (PV) and Persistent Volume Claim (PVC) CRs and patches the **imageregistry** configuration.

NOTE

The local image registry can only be used for user application images and cannot be used for the OpenShift Container Platform or Operator Lifecycle Manager operator images.

Additional resources

- OpenShift Container Platform registry overview

17.9.8.1. Configuring disk partitioning with SiteConfig

Configure disk partitioning for a managed cluster using a **SiteConfig** CR and GitOps ZTP. The disk partition details in the **SiteConfig** CR must match the underlying disk.

NOTE

Use persistent naming for devices to avoid device names such as **/dev/sda** and **/dev/sdb** being switched at every reboot. You can use **rootDeviceHints** to choose the bootable device and then use same device for further partitioning.

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 for use with GitOps Zero Touch Provisioning (ZTP).

Procedure

1. Add the following YAML that describes the host disk partitioning to the **SiteConfig** CR that you use to install the managed cluster:

```
nodes:
    rootDeviceHints:
      wwn: "0x62cea7f05c98c2002708a0a22ff480ea"
    diskPartition:
      - device: /dev/disk/by-id/wwn-0x62cea7f05c98c2002708a0a22ff480ea 1
        partitions:
          - mount_point: /var/imageregistry
            size: 102500 2
            start: 344844 3
```

   1. This setting depends on the hardware. The setting can be a serial number or device name. The value must match the value set for **rootDeviceHints**.

   2. The minimum value for **size** is 102500 MiB.

   3. The minimum value for **start** is 25000 MiB. The total value of **size** and **start** must not exceed the disk size, or the installation will fail.

2. Save the **SiteConfig** CR and push it to the site configuration repo.

The ZTP pipeline provisions the cluster using the **SiteConfig** CR and configures the disk partition.

17.9.8.2. Configuring the image registry using PolicyGenTemplate CRs

Use **PolicyGenTemplate** (PGT) CRs to apply the CRs required to configure the image registry and patch the **imageregistry** configuration.

Prerequisites

- You have configured a disk partition in the managed cluster.
- 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 for use with GitOps Zero Touch Provisioning (ZTP).

Procedure

1. Configure the storage class, persistent volume claim, persistent volume, and image registry configuration in the appropriate **PolicyGenTemplate** CR. For example, to configure an individual site, add the following YAML to the file **example-sno-site.yaml**:

```
sourceFiles:
  # storage class
  - fileName: StorageClass.yaml
    policyName: "sc-for-image-registry"
    metadata:
      name: image-registry-sc
      annotations:
```

```
      ran.openshift.io/ztp-deploy-wave: "100" ❶
# persistent volume claim
- fileName: StoragePVC.yaml
  policyName: "pvc-for-image-registry"
  metadata:
    name: image-registry-pvc
    namespace: openshift-image-registry
    annotations:
      ran.openshift.io/ztp-deploy-wave: "100"
  spec:
    accessModes:
      - ReadWriteMany
    resources:
      requests:
        storage: 100Gi
    storageClassName: image-registry-sc
    volumeMode: Filesystem
# persistent volume
- fileName: ImageRegistryPV.yaml ❷
  policyName: "pv-for-image-registry"
  metadata:
    annotations:
      ran.openshift.io/ztp-deploy-wave: "100"
- fileName: ImageRegistryConfig.yaml
  policyName: "config-for-image-registry"
  complianceType: musthave
  metadata:
    annotations:
      ran.openshift.io/ztp-deploy-wave: "100"
  spec:
    storage:
      pvc:
        claim: "image-registry-pvc"
```

❶ Set the appropriate value for **ztp-deploy-wave** depending on whether you are configuring image registries at the site, common, or group level. **ztp-deploy-wave: "100"** is suitable for development or testing because it allows you to group the referenced source files together.

❷ In **ImageRegistryPV.yaml**, ensure that the **spec.local.path** field is set to **/var/imageregistry** to match the value set for the **mount_point** field in the **SiteConfig** CR.

IMPORTANT

Do not set **complianceType: mustonlyhave** for the **- fileName: ImageRegistryConfig.yaml** configuration. This can cause the registry pod deployment to fail.

2. Commit the **PolicyGenTemplate** change in Git, and then push to the Git repository being monitored by the GitOps ZTP ArgoCD application.

Verification

Use the following steps to troubleshoot errors with the local image registry on the managed clusters:

- Verify successful login to the registry while logged in to the managed cluster. Run the following commands:

  a. Export the managed cluster name:

  ```
  $ cluster=<managed_cluster_name>
  ```

  b. Get the managed cluster **kubeconfig** details:

  ```
  $ oc get secret -n $cluster $cluster-admin-password -o jsonpath='{.data.password}' |
  base64 -d > kubeadmin-password-$cluster
  ```

  c. Download and export the cluster **kubeconfig**:

  ```
  $ oc get secret -n $cluster $cluster-admin-kubeconfig -o jsonpath='{.data.kubeconfig}' |
  base64 -d > kubeconfig-$cluster && export KUBECONFIG=./kubeconfig-$cluster
  ```

  d. Verify access to the image registry from the managed cluster. See "Accessing the registry".

- Check that the **Config** CRD in the **imageregistry.operator.openshift.io** group instance is not reporting errors. Run the following command while logged in to the managed cluster:

  ```
  $ oc get image.config.openshift.io cluster -o yaml
  ```

  Example output

  ```
  apiVersion: config.openshift.io/v1
  kind: Image
  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: "2021-10-08T19:02:39Z"
    generation: 5
    name: cluster
    resourceVersion: "688678648"
    uid: 0406521b-39c0-4cda-ba75-873697da75a4
  spec:
    additionalTrustedCA:
      name: acm-ice
  ```

- Check that the **PersistentVolumeClaim** on the managed cluster is populated with data. Run the following command while logged in to the managed cluster:

  ```
  $ oc get pv image-registry-sc
  ```

- Check that the **registry*** pod is running and is located under the **openshift-image-registry** namespace.

  ```
  $ oc get pods -n openshift-image-registry | grep registry*
  ```

Example output

```
cluster-image-registry-operator-68f5c9c589-42cfg   1/1     Running     0          8d
image-registry-5f8987879-6nx6h                     1/1     Running     0          8d
```

- Check that the disk partition on the managed cluster is correct:

  a. Open a debug shell to the managed cluster:

  ```
  $ oc debug node/sno-1.example.com
  ```

  b. Run **lsblk** to check the host disk partitions:

  ```
  sh-4.4# lsblk
  NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
  sda      8:0    0 446.6G  0 disk
    |-sda1   8:1    0     1M  0 part
    |-sda2   8:2    0   127M  0 part
    |-sda3   8:3    0   384M  0 part /boot
    |-sda4   8:4    0 336.3G  0 part /sysroot
    `-sda5   8:5    0 100.1G  0 part /var/imageregistry 1
  sdb      8:16   0 446.6G  0 disk
  sr0     11:0    1   104M  0 rom
  ```

  1 **/var/imageregistry** indicates that the disk is correctly partitioned.

Additional resources

- Accessing the registry

17.9.9. Using hub templates in PolicyGenTemplate CRs

Topology Aware Lifecycle Manager supports partial Red Hat Advanced Cluster Management (RHACM) hub cluster template functions in configuration policies used with GitOps ZTP.

Hub-side cluster templates allow you to define configuration policies that can be dynamically customized to the target clusters. This reduces the need to create separate policies for many clusters with similiar configurations but with different values.

> **IMPORTANT**
>
> Policy templates are restricted to the same namespace as the namespace where the policy is defined. This means that you must create the objects referenced in the hub template in the same namespace where the policy is created.

The following supported hub template functions are available for use in GitOps ZTP with TALM:

- **fromConfigmap** returns the value of the provided data key in the named **ConfigMap** resource.

NOTE

There is a 1 MiB size limit for **ConfigMap** CRs. The effective size for **ConfigMap** CRs is further limited by the **last-applied-configuration** annotation. To avoid the **last-applied-configuration** limitation, add the following annotation to the template **ConfigMap**:

```
argocd.argoproj.io/sync-options: Replace=true
```

- **base64enc** returns the base64-encoded value of the input string
- **base64dec** returns the decoded value of the base64-encoded input string
- **indent** returns the input string with added indent spaces
- **autoindent** returns the input string with added indent spaces based on the spacing used in the parent template
- **toInt** casts and returns the integer value of the input value
- **toBool** converts the input string into a boolean value, and returns the boolean

Various Open source community functions are also available for use with GitOps ZTP.

Additional resources

- RHACM support for hub cluster templates in configuration policies

17.9.9.1. Example hub templates

The following code examples are valid hub templates. Each of these templates return values from the **ConfigMap** CR with the name **test-config** in the **default** namespace.

- Returns the value with the key **common-key**:

```
{{hub fromConfigMap "default" "test-config" "common-key" hub}}
```

- Returns a string by using the concatenated value of the **.ManagedClusterName** field and the string **-name**:

```
{{hub fromConfigMap "default" "test-config" (printf "%s-name" .ManagedClusterName) hub}}
```

- Casts and returns a boolean value from the concatenated value of the **.ManagedClusterName** field and the string **-name**:

```
{{hub fromConfigMap "default" "test-config" (printf "%s-name" .ManagedClusterName) |
toBool hub}}
```

- Casts and returns an integer value from the concatenated value of the **.ManagedClusterName** field and the string **-name**:

```
{{hub (printf "%s-name" .ManagedClusterName) | fromConfigMap "default" "test-config" | toInt
hub}}
```

17.9.9.2. Specifying host NICs in site PolicyGenTemplate CRs with hub cluster templates

You can manage host NICs in a single **ConfigMap** CR and use hub cluster templates to populate the custom NIC values in the generated polices that get applied to the cluster hosts. Using hub cluster templates in site **PolicyGenTemplate** (PGT) CRs means that you do not need to create multiple single site PGT CRs for each site.

The following example shows you how to use a single **ConfigMap** CR to manage cluster host NICs and apply them to the cluster as polices by using a single **PolicyGenTemplate** site CR.

NOTE

When you use the **fromConfigmap** function, the **printf** variable is only available for the template resource **data** key fields. You cannot use it with **name** and **namespace** fields.

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. The repository must be accessible from the hub cluster and be defined as a source repository for the GitOps ZTP ArgoCD application.

Procedure

1. Create a **ConfigMap** resource that describes the NICs for a group of hosts. For example:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: sriovdata
  namespace: ztp-site
  annotations:
    argocd.argoproj.io/sync-options: Replace=true 1
data:
  example-sno-du_fh-numVfs: "8"
  example-sno-du_fh-pf: ens1f0
  example-sno-du_fh-priority: "10"
  example-sno-du_fh-vlan: "140"
  example-sno-du_mh-numVfs: "8"
  example-sno-du_mh-pf: ens3f0
  example-sno-du_mh-priority: "10"
  example-sno-du_mh-vlan: "150"
```

1 The **argocd.argoproj.io/sync-options** annotation is required only if the **ConfigMap** is larger than 1 MiB in size.

NOTE

The **ConfigMap** must be in the same namespace with the policy that has the hub template substitution.

2. Commit the **ConfigMap** CR in Git, and then push to the Git repository being monitored by the Argo CD application.

3. Create a site PGT CR that uses templates to pull the required data from the **ConfigMap** object. For example:

```
apiVersion: ran.openshift.io/v1
kind: PolicyGenTemplate
metadata:
  name: "site"
  namespace: "ztp-site"
spec:
  remediationAction: inform
  bindingRules:
    group-du-sno: ""
  mcp: "master"
  sourceFiles:
    - fileName: SriovNetwork.yaml
      policyName: "config-policy"
      metadata:
        name: "sriov-nw-du-fh"
      spec:
        resourceName: du_fh
        vlan: '{{hub fromConfigMap "ztp-site" "sriovdata" (printf "%s-du_fh-vlan"
.ManagedClusterName) | toInt hub}}'
    - fileName: SriovNetworkNodePolicy.yaml
      policyName: "config-policy"
      metadata:
        name: "sriov-nnp-du-fh"
      spec:
        deviceType: netdevice
        isRdma: true
        nicSelector:
          pfNames:
          - '{{hub fromConfigMap "ztp-site" "sriovdata" (printf "%s-du_fh-pf"
.ManagedClusterName) | autoindent hub}}'
        numVfs: '{{hub fromConfigMap "ztp-site" "sriovdata" (printf "%s-du_fh-numVfs"
.ManagedClusterName) | toInt hub}}'
        priority: '{{hub fromConfigMap "ztp-site" "sriovdata" (printf "%s-du_fh-priority"
.ManagedClusterName) | toInt hub}}'
        resourceName: du_fh
    - fileName: SriovNetwork.yaml
      policyName: "config-policy"
      metadata:
        name: "sriov-nw-du-mh"
      spec:
        resourceName: du_mh
        vlan: '{{hub fromConfigMap "ztp-site" "sriovdata" (printf "%s-du_mh-vlan"
.ManagedClusterName) | toInt hub}}'
    - fileName: SriovNetworkNodePolicy.yaml
      policyName: "config-policy"
      metadata:
        name: "sriov-nnp-du-mh"
      spec:
        deviceType: vfio-pci
        isRdma: false
        nicSelector:
          pfNames:
          - '{{hub fromConfigMap "ztp-site" "sriovdata" (printf "%s-du_mh-pf"
.ManagedClusterName)  hub}}'
        numVfs: '{{hub fromConfigMap "ztp-site" "sriovdata" (printf "%s-du_mh-numVfs"
.ManagedClusterName) | toInt hub}}'
        priority: '{{hub fromConfigMap "ztp-site" "sriovdata" (printf "%s-du_mh-priority"
.ManagedClusterName) | toInt hub}}'
        resourceName: du_mh
```

4. Commit the site **PolicyGenTemplate** CR in Git and push to the Git repository that is monitored by the ArgoCD application.

NOTE

Subsequent changes to the referenced **ConfigMap** CR are not automatically synced to the applied policies. You need to manually sync the new **ConfigMap** changes to update existing PolicyGenTemplate CRs. See "Syncing new ConfigMap changes to existing PolicyGenTemplate CRs".

17.9.9.3. Specifying VLAN IDs in group PolicyGenTemplate CRs with hub cluster templates

You can manage VLAN IDs for managed clusters in a single **ConfigMap** CR and use hub cluster templates to populate the VLAN IDs in the generated polices that get applied to the clusters.

The following example shows how you how manage VLAN IDs in single **ConfigMap** CR and apply them in individual cluster polices by using a single **PolicyGenTemplate** group CR.

NOTE

When using the **fromConfigmap** function, the **printf** variable is only available for the template resource **data** key fields. You cannot use it with **name** and **namespace** fields.

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. 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 **ConfigMap** CR that describes the VLAN IDs for a group of cluster hosts. For example:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: site-data
  namespace: ztp-group
  annotations:
    argocd.argoproj.io/sync-options: Replace=true 1
```

```
data:
  site-1-vlan: "101"
  site-2-vlan: "234"
```

1. The **argocd.argoproj.io/sync-options** annotation is required only if the **ConfigMap** is larger than 1 MiB in size.

NOTE

The **ConfigMap** must be in the same namespace with the policy that has the hub template substitution.

2. Commit the **ConfigMap** CR in Git, and then push to the Git repository being monitored by the Argo CD application.

3. Create a group PGT CR that uses a hub template to pull the required VLAN IDs from the **ConfigMap** object. For example, add the following YAML snippet to the group PGT CR:

```
- fileName: SriovNetwork.yaml
    policyName: "config-policy"
    metadata:
      name: "sriov-nw-du-mh"
      annotations:
        ran.openshift.io/ztp-deploy-wave: "10"
    spec:
      resourceName: du_mh
      vlan: '{{hub fromConfigMap "" "site-data" (printf "%s-vlan" .ManagedClusterName) | toInt
hub}}'
```

4. Commit the group **PolicyGenTemplate** CR in Git, and then push to the Git repository being monitored by the Argo CD application.

NOTE

Subsequent changes to the referenced **ConfigMap** CR are not automatically synced to the applied policies. You need to manually sync the new **ConfigMap** changes to update existing PolicyGenTemplate CRs. See "Syncing new ConfigMap changes to existing PolicyGenTemplate CRs".

17.9.9.4. Syncing new ConfigMap changes to existing PolicyGenTemplate CRs

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 **PolicyGenTemplate** CR that pulls information from a **ConfigMap** CR using hub cluster templates.

Procedure

1. Update the contents of your **ConfigMap** CR, and apply the changes in the hub cluster.

2. To sync the contents of the updated **ConfigMap** CR to the deployed policy, do either of the following:

   a. Option 1: Delete the existing policy. ArgoCD uses the **PolicyGenTemplate** CR to immediately recreate the deleted policy. For example, run the following command:

   ```
   $ oc delete policy <policy_name> -n <policy_namespace>
   ```

   b. Option 2: Apply a special annotation **policy.open-cluster-management.io/trigger-update** to the policy with a different value every time when you update the **ConfigMap**. For example:

   ```
   $ oc annotate policy <policy_name> -n <policy_namespace> policy.open-cluster-management.io/trigger-update="1"
   ```

   NOTE

   You must apply the updated policy for the changes to take effect. For more information, see Special annotation for reprocessing.

3. Optional: If it exists, delete the **ClusterGroupUpdate** CR that contains the policy. For example:

   ```
   $ oc delete clustergroupupgrade <cgu_name> -n <cgu_namespace>
   ```

   a. Create a new **ClusterGroupUpdate** CR that includes the policy to apply with the updated **ConfigMap** changes. For example, add the following YAML to the file **cgr-example.yaml**:

   ```
   apiVersion: ran.openshift.io/v1alpha1
   kind: ClusterGroupUpgrade
   metadata:
     name: <cgr_name>
     namespace: <policy_namespace>
   spec:
     managedPolicies:
       - <managed_policy>
     enable: true
     clusters:
     - <managed_cluster_1>
     - <managed_cluster_2>
     remediationStrategy:
       maxConcurrency: 2
       timeout: 240
   ```

   b. Apply the updated policy:

   ```
   $ oc apply -f cgr-example.yaml
   ```

17.10. UPDATING MANAGED CLUSTERS WITH THE TOPOLOGY AWARE LIFECYCLE MANAGER

You can use the Topology Aware Lifecycle Manager (TALM) to manage the software lifecycle of multiple clusters. TALM uses Red Hat Advanced Cluster Management (RHACM) policies to perform changes on the target clusters.

17.10.1. 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
- The assignment of a canary cluster

For single-node OpenShift, the Topology Aware Lifecycle Manager (TALM) offers the following features:

- Create a backup of a deployment before an upgrade
- Pre-caching images for clusters with limited bandwidth

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.

17.10.2. 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 CRD" section in "Configuring managed clusters with policies and PolicyGenTemplate resources".

17.10.3. 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 regitry.
- 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.

17.10.4. 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.4.12.x   Topology Aware Lifecycle Manager   4.12.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
```

17.10.5. 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

You can control the start time of an update using the **enable** field in the **ClusterGroupUpgrade** CR. For example, if you have a scheduled maintenance window of four hours, you can prepare a **ClusterGroupUpgrade** CR with the **enable** field set to **false**.

You can set the timeout by configuring the **spec.remediationStrategy.timeout** setting as follows:

```
spec
  remediationStrategy:
          maxConcurrency: 1
          timeout: 240
```

You can use the **batchTimeoutAction** to determine what happens if an update fails for a cluster. You can specify **continue** to skip the failing cluster and continue to upgrade other clusters, or **abort** to stop policy remediation for all clusters. Once the timeout elapses, TALM removes all **enforce** policies to ensure that no further updates are made to clusters.

To apply the changes, you set the **enabled** field to **true**.

For more information see the "Applying update policies to managed clusters" section.

As TALM works through remediation of the policies to the specified clusters, the **ClusterGroupUpgrade** CR can report true or false statuses for a number of conditions.

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

17.10.5.1. Selecting clusters

TALM builds a remediation plan and selects clusters based on the following fields:

- The **clusterLabelSelector** field specifies the labels of the clusters that you want to update. This consists of a list of the standard label selectors from **k8s.io/apimachinery/pkg/apis/meta/v1**. Each selector in the list uses either label value pairs or label expressions. Matches from each selector are added to the final list of clusters along with the matches from the **clusterSelector** field and the **cluster** field.

- 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**, **clusterLabelSelector**, and **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.

Sample **ClusterGroupUpgrade** CR with the enabled **field set to false**

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  creationTimestamp: '2022-11-18T16:27:15Z'
  finalizers:
    - ran.openshift.io/cleanup-finalizer
  generation: 1
  name: talm-cgu
  namespace: talm-namespace
  resourceVersion: '40451823'
  uid: cca245a5-4bca-45fa-89c0-aa6af81a596c
Spec:
  actions:
    afterCompletion:
      deleteObjects: true
    beforeEnable: {}
  backup: false
  clusters: ①
    - spoke1
  enable: false ②
  managedPolicies: ③
    - talm-policy
  preCaching: false
  remediationStrategy: ④
    canaries: ⑤
        - spoke1
    maxConcurrency: 2 ⑥
    timeout: 240
  clusterLabelSelectors: ⑦
    - matchExpressions:
      - key: label1
      operator: In
      values:
        - value1a
        - value1b
  batchTimeoutAction: ⑧
status: ⑨
    computedMaxConcurrency: 2
    conditions:
      - lastTransitionTime: '2022-11-18T16:27:15Z'
        message: All selected clusters are valid
      reason: ClusterSelectionCompleted
      status: 'True'
      type: ClustersSelected (10)
    - lastTransitionTime: '2022-11-18T16:27:15Z'
      message: Completed validation
      reason: ValidationCompleted
      status: 'True'
      type: Validated (11)
    - lastTransitionTime: '2022-11-18T16:37:16Z'
      message: Not enabled
      reason: NotEnabled
      status: 'False'
      type: Progressing
  managedPoliciesForUpgrade:
    - name: talm-policy
      namespace: talm-namespace
  managedPoliciesNs:
    talm-policy: talm-namespace
  remediationPlan:
    - - spoke1
    - - spoke2
      - spoke3
  status:
```

1. Defines the list of clusters to update.
2. The **enable** field is set to **false**.
3. Lists the user-defined set of policies to remediate.
4. Defines the specifics of the cluster updates.
5. Defines the clusters for canary updates.
6. 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.
7. Displays the parameters for selecting clusters.
8. Controls what happens if a batch times out. Possible values are **abort** or **continue**. If unspecified, the default is **continue**.
9. Displays information about the status of the updates.
10. The **ClustersSelected** condition shows that all selected clusters are valid.
11. The **Validated** condition shows that all selected clusters have been validated.

NOTE

Any failures during the update of a canary cluster stops the update process.

When the remediation plan is successfully created, you can you set the **enable** field to **true** and 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 **enable** field of the **ClusterGroupUpgrade** CR is set to **false**.

17.10.5.2. Validating

TALM checks that all specified managed policies are available and correct, and uses the **Validated** condition to report the status and reasons as follows:

- **true**
  Validation is completed.

- **false**
  Policies are missing or invalid, or an invalid platform image has been specified.

17.10.5.3. Pre-caching

Clusters might have limited bandwidth to access the container image registry, which can cause a timeout before the updates are completed. On single-node OpenShift clusters, you can use pre-caching to avoid this. The container image pre-caching starts when you create a **ClusterGroupUpgrade** CR with the **preCaching** field set to **true**.

TALM uses the **PrecacheSpecValid** condition to report status information as follows:

- **true**
  The pre-caching spec is valid and consistent.

- **false**
  The pre-caching spec is incomplete.

TALM uses the **PrecachingSucceeded** condition to report status information as follows:

- **true**
  TALM has concluded the pre-caching process. If pre-caching fails for any cluster, the update fails for that cluster but proceeds for all other clusters. A message informs you if pre-caching has failed for any clusters.

- **false**
  Pre-caching is still in progress for one or more clusters or has failed for all clusters.

For more information see the "Using the container image pre-cache feature" section.

17.10.5.4. Creating a backup

For single-node OpenShift, TALM can create a backup of a deployment before an update. If the update fails, you can recover the previous version and restore a cluster to a working state without requiring a reprovision of applications. To use the backup feature you first create a **ClusterGroupUpgrade** CR with the **backup** field set to **true**. To ensure that the contents of the backup are up to date, the backup is not taken until you set the **enable** field in the **ClusterGroupUpgrade** CR to **true**.

TALM uses the **BackupSucceeded** condition to report the status and reasons as follows:

- **true**
  Backup is completed for all clusters or the backup run has completed but failed for one or more clusters. If backup fails for any cluster, the update fails for that cluster but proceeds for all other clusters.

- **false**
  Backup is still in progress for one or more clusters or has failed for all clusters.

For more information, see the "Creating a backup of cluster resources before upgrade" section.

17.10.5.5. Updating clusters

TALM enforces the policies following the remediation plan. 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.

TALM uses the **Progressing** condition to report the status and reasons as follows:

- **true**
  TALM is remediating non-compliant policies.

- **false**
  The update is not in progress. Possible reasons for this are:

  - All clusters are compliant with all the managed policies.
  - The update has timed out as policy remediation took too long.
  - Blocking CRs are missing from the system or have not yet completed.
  - The **ClusterGroupUpgrade** CR is not enabled.
  - Backup is still in progress.

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. When a cluster complies with the current policy, the next managed policy is applied to it.

Sample ClusterGroupUpgrade CR in the Progressing state

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  creationTimestamp: '2022-11-18T16:27:15Z'
  finalizers:
    - ran.openshift.io/cleanup-finalizer
  generation: 1
  name: talm-cgu
  namespace: talm-namespace
  resourceVersion: '40451823'
  uid: cca245a5-4bca-45fa-89c0-aa6af81a596c
```

```
Spec:
 actions:
  afterCompletion:
   deleteObjects: true
  beforeEnable: {}
 backup: false
 clusters:
  - spoke1
 enable: true
 managedPolicies:
  - talm-policy
 preCaching: true
 remediationStrategy:
  canaries:
    - spoke1
  maxConcurrency: 2
  timeout: 240
 clusterLabelSelectors:
  - matchExpressions:
   - key: label1
   operator: In
   values:
    - value1a
    - value1b
 batchTimeoutAction:
status:
  clusters:
   - name: spoke1
    state: complete
  computedMaxConcurrency: 2
  conditions:
   - lastTransitionTime: '2022-11-18T16:27:15Z'
    message: All selected clusters are valid
    reason: ClusterSelectionCompleted
    status: 'True'
    type: ClustersSelected
   - lastTransitionTime: '2022-11-18T16:27:15Z'
    message: Completed validation
    reason: ValidationCompleted
    status: 'True'
    type: Validated
   - lastTransitionTime: '2022-11-18T16:37:16Z'
    message: Remediating non-compliant policies
    reason: InProgress
    status: 'True'
    type: Progressing 1
  managedPoliciesForUpgrade:
   - name: talm-policy
    namespace: talm-namespace
  managedPoliciesNs:
   talm-policy: talm-namespace
  remediationPlan:
   - - spoke1
   - - spoke2
    - spoke3
  status:
```

```
  currentBatch: 2
  currentBatchRemediationProgress:
    spoke2:
      state: Completed
    spoke3:
      policyIndex: 0
      state: InProgress
  currentBatchStartedAt: '2022-11-18T16:27:16Z'
  startedAt: '2022-11-18T16:27:15Z'
```

① The **Progressing** fields show that TALM is in the process of remediating policies.

17.10.5.6. Update status

TALM uses the **Succeeded** condition to report the status and reasons as follows:

- **true**
  All clusters are compliant with the specified managed policies.

- **false**
  Policy remediation failed as there were no clusters available for remediation, or because policy remediation took too long for one of the following reasons:

  - The current batch contains canary updates and the cluster in the batch does not comply with all the managed policies within the batch timeout.

  - Clusters did not comply with the managed policies within the **timeout** value specified in the **remediationStrategy** field.

Sample ClusterGroupUpgrade CR in the Succeeded state

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-upgrade-complete
  namespace: default
spec:
  clusters:
  - spoke1
  - spoke4
  enable: true
  managedPolicies:
  - policy1-common-cluster-version-policy
  - policy2-common-pao-sub-policy
  remediationStrategy:
    maxConcurrency: 1
    timeout: 240
status: ①
  clusters:
    - name: spoke1
      state: complete
    - name: spoke4
      state: complete
  conditions:
      - message: All selected clusters are valid
        reason: ClusterSelectionCompleted
        status: "True"
        type: ClustersSelected
      - message: Completed validation
        reason: ValidationCompleted
        status: "True"
        type: Validated
      - message: All clusters are compliant with all the managed policies
        reason: Completed
        status: "False"
        type: Progressing ②
      - message: All clusters are compliant with all the managed policies
        reason: Completed
        status: "True"
        type: Succeeded ③
      managedPoliciesForUpgrade:
      - name: policy1-common-cluster-version-policy
        namespace: default
      - name: policy2-common-pao-sub-policy
        namespace: default
      remediationPlan:
      - - spoke1
      - - spoke4
      status:
        completedAt: '2022-11-18T16:27:16Z'
        startedAt: '2022-11-18T16:27:15Z'
```

② In the **Progressing** fields, the status is **false** as the update has completed; clusters are compliant with all the managed policies.

③ The **Succeeded** fields show that the validations completed successfully.

① The **status** field includes a list of clusters and their respective statuses. The status of a cluster can be **complete** or **timedout**.

Sample ClusterGroupUpgrade CR in the timedout state

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  creationTimestamp: '2022-11-18T16:27:15Z'
  finalizers:
    - ran.openshift.io/cleanup-finalizer
  generation: 1
  name: talm-cgu
  namespace: talm-namespace
  resourceVersion: '40451823'
  uid: cca245a5-4bca-45fa-89c0-aa6af81a596c
spec:
  actions:
    afterCompletion:
      deleteObjects: true
    beforeEnable: {}
```

```
 backup: false
 clusters:
  - spoke1
  - spoke2
 enable: true
 managedPolicies:
  - talm-policy
 preCaching: false
 remediationStrategy:
  maxConcurrency: 2
  timeout: 240
status:
 clusters:
  - name: spoke1
   state: complete
  - currentPolicy: 1
     name: talm-policy
     status: NonCompliant
   name: spoke2
   state: timedout
 computedMaxConcurrency: 2
 conditions:
  - lastTransitionTime: '2022-11-18T16:27:15Z'
   message: All selected clusters are valid
   reason: ClusterSelectionCompleted
   status: 'True'
   type: ClustersSelected
  - lastTransitionTime: '2022-11-18T16:27:15Z'
   message: Completed validation
   reason: ValidationCompleted
   status: 'True'
   type: Validated
  - lastTransitionTime: '2022-11-18T16:37:16Z'
   message: Policy remediation took too long
   reason: TimedOut
   status: 'False'
   type: Progressing
  - lastTransitionTime: '2022-11-18T16:37:16Z'
   message: Policy remediation took too long
   reason: TimedOut
   status: 'False'
   type: Succeeded 2
 managedPoliciesForUpgrade:
  - name: talm-policy
   namespace: talm-namespace
 managedPoliciesNs:
  talm-policy: talm-namespace
 remediationPlan:
  - - spoke1
   - spoke2
 status:
     startedAt: '2022-11-18T16:27:15Z'
     completedAt: '2022-11-18T20:27:15Z'
```

1 If a cluster’s state is **timedout**, the **currentPolicy** field shows the name of the policy and the policy status.

2 The status for **succeeded** is **false** and the message indicates that policy remediation took too long.

17.10.5.7. 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: 1
  - name: cgu-c
    namespace: default
  clusters:
  - spoke1
  - spoke2
  - spoke3
  enable: false
  managedPolicies:
  - policy1-common-cluster-version-policy
  - policy2-common-pao-sub-policy
  - policy3-common-ptp-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-ptp-sub-policy
  managedPoliciesForUpgrade:
  - name: policy1-common-cluster-version-policy
    namespace: default
  - name: policy2-common-pao-sub-policy
    namespace: default
  - name: policy3-common-ptp-sub-policy
    namespace: default
  placementBindings:
  - cgu-a-policy1-common-cluster-version-policy
  - cgu-a-policy2-common-pao-sub-policy
  - cgu-a-policy3-common-ptp-sub-policy
  placementRules:
  - cgu-a-policy1-common-cluster-version-policy
  - cgu-a-policy2-common-pao-sub-policy
  - cgu-a-policy3-common-ptp-sub-policy
  remediationPlan:
  - - spoke1
  - - spoke2
```

1 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: 1
  - name: cgu-a
    namespace: default
  clusters:
  - spoke4
  - spoke5
  enable: false
  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 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-ptp-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-ptp-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-ptp-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-ptp-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: 1
  clusters:
  - spoke6
  enable: false
  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 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-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: {}
```

1 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-ptp-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-ptp-sub-policy
  managedPoliciesForUpgrade:
  - name: policy1-common-cluster-version-policy
    namespace: default
  - name: policy2-common-pao-sub-policy
    namespace: default
  - name: policy3-common-ptp-sub-policy
    namespace: default
  placementBindings:
  - cgu-a-policy1-common-cluster-version-policy
  - cgu-a-policy2-common-pao-sub-policy
  - cgu-a-policy3-common-ptp-sub-policy
  placementRules:
  - cgu-a-policy1-common-cluster-version-policy
  - cgu-a-policy2-common-pao-sub-policy
  - cgu-a-policy3-common-ptp-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-ptp-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-ptp-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-ptp-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-ptp-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-ptp-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.

17.10.6. 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 noncompliant 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.

The **ClusterGroupUpgrade** CR's **batchTimeoutAction** determines what happens if an upgrade fails for a cluster. You can specify **continue** to skip the failing cluster and continue to upgrade other clusters, or specify **abort** to stop the policy remediation for all clusters. Once the timeout elapses, TALM removes all enforce policies to ensure that no further updates are made to clusters.

Example upgrade policy

```
apiVersion: policy.open-cluster-management.io/v1
kind: Policy
metadata:
  name: ocp-4.4.12.4
  namespace: platform-upgrade
spec:
  disabled: false
  policy-templates:
  - objectDefinition:
      apiVersion: policy.open-cluster-management.io/v1
      kind: ConfigurationPolicy
      metadata:
        name: upgrade
      spec:
        namespaceselector:
          exclude:
          - kube-*
          include:
          - '*'
        object-templates:
        - complianceType: musthave
          objectDefinition:
            apiVersion: config.openshift.io/v1
            kind: ClusterVersion
            metadata:
              name: version
            spec:
              channel: stable-4.12
              desiredUpdate:
                version: 4.4.12.4
              upstream: https://api.openshift.com/api/upgrades_info/v1/graph
            status:
              history:
                - state: Completed
                version: 4.4.12.4
      remediationAction: inform
      severity: low
remediationAction: inform
```

For more information about RHACM policies, see Policy overview.

Additional resources

For more information about the **PolicyGenTemplate** CRD, see About the PolicyGenTemplate CRD.

17.10.6.1. Configuring Operator subscriptions for managed clusters that you install with TALM

Topology Aware Lifecycle Manager (TALM) can only approve the install plan for an Operator if the **Subscription** custom resource (CR) of the Operator contains the **status.state.AtLatestKnown** field.

Procedure

1. Add the **status.state.AtLatestKnown** field to the **Subscription** CR of the Operator:

   Example Subscription CR

   ```
   apiVersion: operators.coreos.com/v1alpha1
   kind: Subscription
   metadata:
     name: cluster-logging
     namespace: openshift-logging
     annotations:
       ran.openshift.io/ztp-deploy-wave: "2"
   spec:
     channel: "stable"
     name: cluster-logging
     source: redhat-operators
     sourceNamespace: openshift-marketplace
     installPlanApproval: Manual
   status:
     state: AtLatestKnown (1)
   ```

   (1) The **status.state: AtLatestKnown** field is used for the latest Operator version available from the Operator catalog.

   NOTE

   When a new version of the Operator is available in the registry, the associated policy becomes non-compliant.

2. Apply the changed **Subscription** policy to your managed clusters with a **ClusterGroupUpgrade** CR.

17.10.6.2. 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
  batchTimeoutAction: ❺
```

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.

❺ Controls what happens if a batch times out. Possible values are **abort** or **continue**. If unspecified, the default is **continue**.

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  STATE      DETAILS
default     cgu-1 8m55 NotEnabled Not Enabled
```

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": "Not enabled", 1
      "reason": "NotEnabled",
      "status": "False",
      "type": "Progressing"
    }
  ],
  "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-ptp-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                                          REMEDIATION ACTION
COMPLIANCE STATE   AGE
default     cgu-policy1-common-cluster-version-policy        enforce
17m 1
default     cgu-policy2-common-nto-sub-policy                enforce
17m
```

```
default    cgu-policy3-common-ptp-sub-policy                  enforce
17m
default    cgu-policy4-common-sriov-sub-policy                enforce
17m
default    policy1-common-cluster-version-policy              inform          NonCompliant
15h
default    policy2-common-nto-sub-policy                    inform          NonCompliant
15h
default    policy3-common-ptp-sub-policy                    inform          NonCompliant
18m
default    policy4-common-sriov-sub-policy                  inform          NonCompliant
18m
```

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": "2022-02-25T15:33:07Z",
      "message": "All selected clusters are valid",
      "reason": "ClusterSelectionCompleted",
      "status": "True",
      "type": "ClustersSelected",
      "lastTransitionTime": "2022-02-25T15:33:07Z",
      "message": "Completed validation",
      "reason": "ValidationCompleted",
      "status": "True",
      "type": "Validated",
      "lastTransitionTime": "2022-02-25T15:34:07Z",
      "message": "Remediating non-compliant policies",
      "reason": "InProgress",
      "status": "True",
      "type": "Progressing"
    }
  ],
  "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-ptp-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   4.4.12.5     True        True          43s     Working towards 4.4.12.7: 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
```

1 The install plans have their **Approval** field set to **Manual** and their **Approved** field changes from **false** to **true** after TALM approves the install plan.

NOTE

When TALM is remediating a policy containing a subscription, it automatically approves any install plans attached to that subscription. Where multiple install plans are needed to get the operator to the latest known version, TALM might approve multiple install plans, upgrading through one or more intermediate versions to get to the final version.

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
```

17.10.7. 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.

To use the backup feature you first create a **ClusterGroupUpgrade** CR with the **backup** field set to **true**. To ensure that the contents of the backup are up to date, the backup is not taken until you set the **enable** field in the **ClusterGroupUpgrade** CR to **true**.

TALM uses the **BackupSucceeded** condition to report the status and reasons as follows:

- **true**
  Backup is completed for all clusters or the backup run has completed but failed for one or more clusters. If backup fails for any cluster, the update does not proceed for that cluster.

- **false**
  Backup is still in progress for one or more clusters or has failed for all clusters. The backup process running in the spoke clusters can have the following statuses:
  - **PreparingToStart**
    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.
  - **Starting**
    The backup prerequisites and backup job are being created.
  - **Active**
    The backup is in progress.
  - **Succeeded**
    The backup succeeded.
  - **BackupTimeout**
    Artifact backup is partially done.
  - **UnrecoverableError**
    The backup has ended with a non-zero exit code.

NOTE

If the backup of a cluster fails and enters the **BackupTimeout** or **UnrecoverableError** state, the cluster update does not proceed for that cluster. Updates to other clusters are not affected and continue.

17.10.7.1. Creating a ClusterGroupUpgrade CR with backup

You can create a backup of a deployment before an upgrade on single-node OpenShift clusters. 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: "node-1.example.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** and **enable** fields 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: true
  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": "Failed" ①
   }
},
"computedMaxConcurrency": 1,
"conditions": [
   {
      "lastTransitionTime": "2022-04-05T10:37:19Z",
      "message": "Backup failed for 1 cluster", ②
      "reason": "PartiallyDone", ③
      "status": "True", ④
      "type": "Succeeded"
   }
],
"precaching": {
   "spec": {}
},
"status": {}
```

① Backup has failed for one cluster.

② The message confirms that the backup failed for one cluster.

③ The backup was partially successful.

④ The backup process has finished.

17.10.7.2. 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.

IMPORTANT

A rollback is only applicable to upgrades from TALM and single-node OpenShift. This process does not apply to rollbacks from any other upgrade type.

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:

```
$ ostree admin status
```

Example outputs

```
[root@lab-test-spoke2-node-0 core]# ostree admin status
* rhcos c038a8f08458bbed83a77ece033ad3c55597e3f64edad66ea12fda18cbdceaf9.0
    Version: 49.84.202202230006-0
    Pinned: yes 1
    origin refspec:
c038a8f08458bbed83a77ece033ad3c55597e3f64edad66ea12fda18cbdceaf9
```

1 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 refspec: f750ff26f2d5550930ccbe17af61af47daafc8018cd9944f2a3a6269af26b0fa
rhcos ad8f159f9dc4ea7e773fd9604c9a16be0fe9b266ae800ac8470f63abc39b52ca.0
(rollback) 1
    Version: 410.84.202203290245-0
    Pinned: yes 2
    origin refspec:
ad8f159f9dc4ea7e773fd9604c9a16be0fe9b266ae800ac8470f63abc39b52ca
```

1 This platform OS deployment is marked for rollback.

2 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
```

5. 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
```

6. When prompted, reboot the cluster by running the following command:

```
$ systemctl reboot
```

7. 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   4.4.12.23    True        False        86d     Cluster
version is 4.4.12.23 1


NAME                       STATUS   ROLES          AGE   VERSION
node/lab-test-spoke1-node-0   Ready    master,worker   86d   v1.22.3+b93fd35 2

NAME                                                     VERSION   AVAILABLE
PROGRESSING   DEGRADED   SINCE   MESSAGE
clusteroperator.config.openshift.io/authentication                    4.4.12.23    True
False        False      2d7h  3
clusteroperator.config.openshift.io/baremetal                        4.4.12.23    True        False
False      86d


..............
```

1. The cluster version is available and has the correct version.
2. The node status is **Ready**.
3. The **ClusterOperator** object’s availability is **True**.

17.10.8. Using the container image pre-cache feature

Single-node OpenShift 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.

TALM uses the **PrecacheSpecValid** condition to report status information as follows:

- **true**
  The pre-caching spec is valid and consistent.
- **false**
  The pre-caching spec is incomplete.

TALM uses the **PrecachingSucceeded** condition to report status information as follows:

- **true**
  TALM has concluded the pre-caching process. If pre-caching fails for any cluster, the update fails for that cluster but proceeds for all other clusters. A message informs you if pre-caching has failed for any clusters.
- **false**
  Pre-caching is still in progress for one or more clusters or has failed for all clusters.

After a successful pre-caching process, you can start remediating policies. The remediation actions start when the **enable** field is set to **true**. If there is a pre-caching failure on a cluster, the upgrade fails for that cluster. The upgrade process continues for all other clusters that have a successful pre-cache.

The pre-caching process can be in the following statuses:

- **NotStarted**
  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.
- **PreparingToStart**
  Cleaning up any remaining resources from previous incomplete updates is in progress.
- **Starting**

Pre-caching job prerequisites and the job are created.

- **Active**
  The job is in "Active" state.

- **Succeeded**
  The pre-cache job succeeded.

- **PrecacheTimeout**
  The artifact pre-caching is partially done.

- **UnrecoverableError**
  The job ends with a non-zero exit code.

17.10.8.1. Creating a ClusterGroupUpgrade CR with pre-caching

For single-node OpenShift, 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 pre-caching, 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   STATE        DETAILS
ztp-group-du-sno   du-upgrade-4918   10s   InProgress   Precaching is required and not done
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 required and not done",
      "reason": "InProgress",
      "status": "False",
      "type": "PrecachingSucceeded"
    },
    {
      "lastTransitionTime": "2022-01-27T19:07:34Z",
      "message": "Pre-caching spec is valid and consistent",
      "reason": "PrecacheSpecIsWellFormed",
      "status": "True",
      "type": "PrecacheSpecValid"
    }
  ],
  "precaching": {
    "clusters": [
      "cnfdb1" 1
      "cnfdb2"
    ],
    "spec": {
      "platformImage": "image.example.io"},
    "status": {
      "cnfdb1": "Active"
      "cnfdb2": "Succeeded"}
    }
}
```

1 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-talo-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": "PrecachingSucceeded" 1
    }
```

1 The pre-cache tasks are done.

17.10.9. 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

17.10.9.1. 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 17.10.9.3, “Managed policies”
  - Section 17.10.9.4, “Clusters”
  - Section 17.10.9.5, “Remediation Strategy”
  - Section 17.10.9.6, “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.

17.10.9.2. 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>
```

17.10.9.3. 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
```

17.10.9.4. 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
AGE
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          HUB ACCEPTED   MANAGED CLUSTER URLS                  JOINED   AVAILABLE
AGE
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 clusterLabelSelector

Issue

You want to check if the **clusterLabelSelector** field specified in the **ClusterGroupUpgrade** CR matches 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          HUB ACCEPTED   MANAGED CLUSTER URLS                  JOINED
AVAILABLE   AGE
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:
    remediationStrategy:
        canaries:
        - spoke3
        maxConcurrency: 2
        timeout: 240
    clusterLabelSelectors:
      - matchLabels:
          upgrade: true
```

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 **clusterLabelSelector** labels by running the following command:

   ```
   $ oc get managedcluster --selector=upgrade=true
   ```

   #### Example output

   ```
   NAME         HUB ACCEPTED   MANAGED CLUSTER URLS   JOINED    AVAILABLE
   AGE
   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.clusterLabelSelector** 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
   ```

17.10.9.5. 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
```

17.10.9.6. 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":"Missing managed policies:[policyList]",
"reason":"NotAllManagedPoliciesExist", "status":"False", "type":"Validated"}
```

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-ptp-sub-policy
  managedPoliciesForUpgrade:
  - name: policy3-common-ptp-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.

Clusters are not compliant to some policies after a**ClusterGroupUpgrade** CR has completed

Issue

The policy compliance status that TALM uses to decide if remediation is needed has not yet fully updated for all clusters. This may be because:

- The CGU was run too soon after a policy was created or updated.
- The remediation of a policy affects the compliance of subsequent policies in the **ClusterGroupUpgrade** CR.

Resolution

Create a new and apply **ClusterGroupUpdate** CR with the same specification .

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 .
- For more information about the **PolicyGenTemplate** CRD, see About the PolicyGenTemplate CRD

17.11. UPDATING MANAGED CLUSTERS IN A DISCONNECTED ENVIRONMENT WITH THE TOPOLOGY AWARE LIFECYCLE MANAGER

You can use the Topology Aware Lifecycle Manager (TALM) to manage the software lifecycle of OpenShift Container Platform managed clusters. TALM uses Red Hat Advanced Cluster Management (RHACM) policies to perform changes on the target clusters.

Additional resources

- For more information about the Topology Aware Lifecycle Manager, see About the Topology Aware Lifecycle Manager.

17.11.1. Updating clusters in a disconnected environment

You can upgrade managed clusters and Operators for managed clusters that you have deployed using GitOps ZTP and Topology Aware Lifecycle Manager (TALM).

17.11.1.1. 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 managed cluster. To download the update graph, use the following command:

   ```
   $ curl -s https://api.openshift.com/api/upgrades_info/v1/graph?channel=stable-4.12 -
   o ~/upgrade-graph_stable-4.12
   ```

- 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 update channels and releases.

17.11.1.2. 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"
      metadata:
        name: version
      spec:
        channel: "stable-4.12"
        upstream: http://upgrade.example.com/images/upgrade-graph_stable-4.12
        desiredUpdate:
          version: 4.12.4
      status:
        history:
          - version: 4.12.4
            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

4 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 managed 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. 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-prep** and the **du-upgrade-platform-upgrade** policies 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-prep
  - 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
```

3. 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}'
```

4. 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, see Preparing the disconnected environment.

17.11.1.3. 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:v4.12 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 4.12 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 managed 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 1
  - common-subscriptions-policy 2
  preCaching: false
  clusters:
  - spoke1
  remediationStrategy:
    maxConcurrency: 1
  enable: false
```

1 The policy is needed by the image pre-caching feature to retrieve the operator images from the catalog source.

2 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 managed 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 .
- Troubleshooting missed Operator updates due to out-of-date policy compliance states .

17.11.1.3.1. Troubleshooting missed Operator updates due to out-of-date policy compliance states

In some scenarios, Topology Aware Lifecycle Manager (TALM) might miss Operator updates due to an out-of-date policy compliance state.

After a catalog source update, it takes time for the Operator Lifecycle Manager (OLM) to update the subscription status. The status of the subscription policy might continue to show as compliant while TALM decides whether remediation is needed. As a result, the Operator specified in the subscription policy does not get upgraded.

To avoid this scenario, add another catalog source configuration to the **PolicyGenTemplate** and specify this configuration in the subscription for any Operators that require an update.

Procedure

1. Add a catalog source configuration in the **PolicyGenTemplate** resource:

```
- 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:v{product-version}
      updateStrategy:
        registryPoll:
```

```
            interval: 1h
      status:
        connectionState:
            lastObservedState: READY
- fileName: DefaultCatsrc.yaml
      remediationAction: inform
      policyName: "operator-catsrc-policy"
      metadata:
        name: redhat-operators-v2 1
      spec:
        displayName: Red Hat Operators Catalog v2 2
        image: registry.example.com:5000/olredhat-operators:<version> 3
        updateStrategy:
          registryPoll:
            interval: 1h
      status:
        connectionState:
            lastObservedState: READY
```

1 Update the name for the new configuration.

2 Update the display name for the new configuration.

3 Update the index image URL. This **fileName.spec.image** field overrides any configuration in the **DefaultCatsrc.yaml** file.

2. Update the **Subscription** resource to point to the new configuration for Operators that require an update:

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: operator-subscription
  namespace: operator-namspace
# ...
spec:
  source: redhat-operators-v2 1
# ...
```

1 Enter the name of the additional catalog source configuration that you defined in the **PolicyGenTemplate** resource.

17.11.1.4. 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 1
   ```

```
- du-upgrade-operator-catsrc-policy 2
- common-subscriptions-policy 3
preCaching: true
clusterSelector:
- group-du-sno
remediationStrategy:
  maxConcurrency: 1
enable: false
```

1 This is the platform update policy.

2 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 managed cluster.

3 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 managed 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.

17.11.1.5. 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.

NOTE

You need to remove any policies that create Performance Addon Operator subscriptions to prevent a re-installation of the Operator.

The reference DU profile includes the Performance Addon Operator in the **PolicyGenTemplate** CR **common-ranGen.yaml**. To remove the subscription from deployed managed 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 ArgoCD.
- 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 ztp-common 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

- For more information about the TALM pre-caching workflow, see Using the container image pre-cache feature.

17.11.2. 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: "" 1
      deleteClusterLabels:
        ztp-running: ""
      deleteObjects: true
    beforeEnable:
      addClusterLabels:
        ztp-running: "" 2
  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
```

1 Applied to the managed cluster when TALM completes the cluster configuration.

2 Applied to the managed cluster when TALM starts deploying the configuration policies.

17.12. UPDATING GITOPS ZTP

You can update the Gitops zero touch provisioning (ZTP) infrastructure independently from the hub cluster, Red Hat Advanced Cluster Management (RHACM), and the managed OpenShift Container Platform clusters.

NOTE

You can update the Red Hat OpenShift GitOps Operator when new versions become available. When updating the GitOps ZTP plugin, review the updated files in the reference configuration and ensure that the changes meet your requirements.

17.12.1. Overview of the GitOps ZTP update process

You can update GitOps zero touch provisioning (ZTP) for a fully operational hub cluster running an earlier version of the GitOps ZTP infrastructure. The update process avoids impact on managed clusters.

> **NOTE**
>
> Any changes to policy settings, including adding recommended content, results in updated polices that must be rolled out to the managed clusters and reconciled.

At a high level, the strategy for updating the GitOps ZTP infrastructure is as follows:

1. Label all existing clusters with the **ztp-done** label.
2. Stop the ArgoCD applications.
3. Install the new GitOps ZTP tools.
4. Update required content and optional changes in the Git repository.
5. Update and restart the application configuration.

17.12.2. Preparing for the upgrade

Use the following procedure to prepare your site for the GitOps zero touch provisioning (ZTP) upgrade.

Procedure

1. Get the latest version of the GitOps ZTP container that has the custom resources (CRs) used to configure Red Hat OpenShift GitOps for use with GitOps ZTP.
2. Extract the **argocd/deployment** directory by using the following commands:

   ```
   $ mkdir -p ./update
   ```

   ```
   $ podman run --log-driver=none --rm registry.redhat.io/openshift4/ztp-site-generate-rhel8:v4.12 extract /home/ztp --tar | tar x -C ./update
   ```

   The **/update** directory contains the following subdirectories:

   - **update/extra-manifest**: contains the source CR files that the **SiteConfig** CR uses to generate the extra manifest **configMap**.
   - **update/source-crs**: contains the source CR files that the **PolicyGenTemplate** CR uses to generate the Red Hat Advanced Cluster Management (RHACM) policies.
   - **update/argocd/deployment**: contains patches and YAML files to apply on the hub cluster for use in the next step of this procedure.
   - **update/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 that results in obsolete policies, the obsolete policies should be removed prior to performing the upgrade.

4. Diff the changes between the configuration and deployment source CRs in the /**update** folder and Git repo where you manage your fleet site CRs. Apply and push the required changes to your site repository.

> IMPORTANT
>
> When you update GitOps ZTP to the latest version, you must apply the changes from the **update/argocd/deployment** directory to your site repository. Do not use older versions of the **argocd/deployment/** files.

17.12.3. Labeling the existing clusters

To ensure that existing clusters remain untouched by the tool updates, label all existing managed clusters with the **ztp-done** label.

> NOTE
>
> This procedure only applies when updating clusters that were not provisioned with Topology Aware Lifecycle Manager (TALM). Clusters that you provision with TALM are automatically labeled with **ztp-done**.

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=
```

17.12.4. 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 tools 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 update/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 update/argocd/deployment/policies-app.yaml
```

17.12.5. Required changes to the Git repository

When upgrading the **ztp-site-generate** container from an earlier release of GitOps ZTP to v4.10 or later, there are additional requirements for the contents of the Git repository. Existing content in the repository must be updated to reflect these changes.

- Make required 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.

- 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:

```
├── policygentemplates
│   ├── site1-ns.yaml
│   ├── site1.yaml
│   ├── site2-ns.yaml
│   ├── site2.yaml
│   ├── common-ns.yaml
│   ├── common-ranGen.yaml
│   ├── group-du-sno-ranGen-ns.yaml
│   ├── group-du-sno-ranGen.yaml
│   └── kustomization.yaml
└── siteconfig
    ├── site1.yaml
    ├── site2.yaml
    └── kustomization.yaml
```

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
```

- Remove the **pre-sync.yaml** and **post-sync.yaml** files.
  In OpenShift Container Platform 4.10 and later, the **pre-sync.yaml** and **post-sync.yaml** files are no longer required. The **update/deployment/kustomization.yaml** CR manages the policies deployment on the hub cluster.

  NOTE

  There is a set of **pre-sync.yaml** and **post-sync.yaml** files under both the **SiteConfig** and **PolicyGenTemplate** trees.

- 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.

17.12.6. Installing the new GitOps ZTP applications

Using the extracted **argocd/deployment** directory, and after ensuring that the applications point to your site 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 that you previously extracted into the **update/argocd/deployment/** directory, enter the following command:

```
$ oc patch argocd openshift-gitops \
-n openshift-gitops --type=merge \
--patch-file update/argocd/deployment/argocd-openshift-gitops-patch.json
```

2. To apply the contents of the **argocd/deployment** directory, enter the following command:

```
$ oc apply -k update/argocd/deployment
```

17.12.7. Rolling out the GitOps ZTP 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. With the ZTP GitOps v4.10 and later **ztp-site-generate** container, these policies are set to **inform** mode and are not pushed to the managed 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 managed clusters as well as a list or selector of which clusters should be included in the update.

Additional resources

- For information about the Topology Aware Lifecycle Manager (TALM), see About the Topology Aware Lifecycle Manager configuration.
- For information about creating **ClusterGroupUpgrade** CRs, see About the auto-created ClusterGroupUpgrade CR for ZTP.

17.13. EXPANDING SINGLE-NODE OPENSHIFT CLUSTERS WITH GITOPS ZTP

You can expand single-node OpenShift clusters with GitOps ZTP. When you add worker nodes to single-node OpenShift clusters, the original single-node OpenShift cluster retains the control plane node role. Adding worker nodes does not require any downtime for the existing single-node OpenShift cluster.

NOTE

Although there is no specified limit on the number of worker nodes that you can add to a single-node OpenShift cluster, you must revaluate the reserved CPU allocation on the control plane node for the additional worker nodes.

If you require workload partitioning on the worker node, you must deploy and remediate the managed cluster policies on the hub cluster before installing the node. This way, the workload partitioning **MachineConfig** objects are rendered and associated with the **worker** machine config pool before the GitOps ZTP workflow applies the **MachineConfig** ignition file to the worker node.

It is recommended that you first remediate the policies, and then install the worker node. If you create the workload partitioning manifests after installing the worker node, you must drain the node manually and delete all the pods managed by daemon sets. When the managing daemon sets create the new pods, the new pods undergo the workload partitioning process.

IMPORTANT

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

For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

Additional resources

- For more information about single-node OpenShift clusters tuned for vDU application deployments, see Reference configuration for deploying vDUs on single-node OpenShift .
- For more information about worker nodes, see Adding worker nodes to single-node OpenShift clusters.

17.13.1. Applying profiles to the worker node

You can configure the additional worker node with a DU profile.

You can apply a RAN distributed unit (DU) profile to the worker node cluster using the ZTP GitOps common, group, and site-specific **PolicyGenTemplate** resources. The GitOps ZTP pipeline that is linked to the ArgoCD **policies** application includes the following CRs that you can find in the **out/argocd/example/policygentemplates** folder when you extract the **ztp-site-generate** container:

- **common-ranGen.yaml**
- **group-du-sno-ranGen.yaml**
- **example-sno-site.yaml**
- **ns.yaml**
- **kustomization.yaml**

Configuring the DU profile on the worker node is considered an upgrade. To initiate the upgrade flow, you must update the existing policies or create additional ones. Then, you must create a **ClusterGroupUpgrade** CR to reconcile the policies in the group of clusters.

17.13.2. (Optional) Ensuring PTP and SR-IOV daemon selector compatibility

If the DU profile was deployed using the GitOps ZTP plugin version 4.11 or earlier, the PTP and SR-IOV Operators might be configured to place the daemons only on nodes labelled as **master**. This configuration prevents the PTP and SR-IOV daemons from operating on the worker node. If the PTP and SR-IOV daemon node selectors are incorrectly configured on your system, you must change the daemons before proceeding with the worker DU profile configuration.

Procedure

1. Check the daemon node selector settings of the PTP Operator on one of the spoke clusters:

```
$ oc get ptpoperatorconfig/default -n openshift-ptp -ojsonpath='{.spec}' | jq
```

Example output for PTP Operator

```
{"daemonNodeSelector":{"node-role.kubernetes.io/master":""}} 1
```

1 If the node selector is set to **master**, the spoke was deployed with the version of the ZTP plugin that requires changes.

2. Check the daemon node selector settings of the SR-IOV Operator on one of the spoke clusters:

```
$ oc get sriovoperatorconfig/default -n \
openshift-sriov-network-operator -ojsonpath='{.spec}' | jq
```

Example output for SR-IOV Operator

```
{"configDaemonNodeSelector":{"node-
role.kubernetes.io/worker":""},"disableDrain":false,"enableInjector":true,"enableOperatorWebh
ook":true} 1
```

1 If the node selector is set to **master**, the spoke was deployed with the version of the ZTP plugin that requires changes.

3. In the group policy, add the following **complianceType** and **spec** entries:

```
spec:
    - fileName: PtpOperatorConfig.yaml
      policyName: "config-policy"
      complianceType: mustonlyhave
      spec:
        daemonNodeSelector:
          node-role.kubernetes.io/worker: ""
    - fileName: SriovOperatorConfig.yaml
      policyName: "config-policy"
      complianceType: mustonlyhave
      spec:
        configDaemonNodeSelector:
          node-role.kubernetes.io/worker: ""
```

IMPORTANT

Changing the **daemonNodeSelector** field causes temporary PTP synchronization loss and SR-IOV connectivity loss.

4. Commit the changes in Git, and then push to the Git repository being monitored by the GitOps ZTP ArgoCD application.

17.13.3. PTP and SR-IOV node selector compatibility

The PTP configuration resources and SR-IOV network node policies use **node-role.kubernetes.io/master: ""** as the node selector. If the additional worker nodes have the same NIC configuration as the control plane node, the policies used to configure the control plane node can be reused for the worker nodes. However, the node selector must be changed to select both node types, for example with the **"node-role.kubernetes.io/worker"** label.

17.13.4. Using PolicyGenTemplate CRs to apply worker node policies to worker nodes

You can create policies for worker nodes.

Procedure

1. Create the following policy template:

```
apiVersion: ran.openshift.io/v1
kind: PolicyGenTemplate
metadata:
  name: "example-sno-workers"
  namespace: "example-sno"
spec:
  bindingRules:
    sites: "example-sno" 1
  mcp: "worker" 2
  sourceFiles:
    - fileName: MachineConfigGeneric.yaml 3
      policyName: "config-policy"
      metadata:
        labels:
          machineconfiguration.openshift.io/role: worker
        name: enable-workload-partitioning
      spec:
        config:
          storage:
            files:
            - contents:
                source: data:text/plain;charset=utf-
8;base64,W2NyaW8ucnVudGltZS53b3JrbG9hZHMubWFuYWdlbWVudF0KYWN0aXZhdGlvbl
9hbm5vdGF0aW9uID0gInRhcmdldC53b3JrbG9hZC5vcGVuc2hpZnQuaW8vbWFuYWdlbWVu
dCIKYW5ub3RhdGlvbl9wcmVmaXggPSAicmVzb3VyY2VzLndvcmtsb2FkLm9wZW5zaGlmdC5
pbyIKcmVzb3VyY2VzID0geyAiY3B1c2hhcmVzIiA9IDAsICJjcHVzZXQiID0gIjAtMyIgfQo=
              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,ewogICJtYW5hZ2VtZW50IjogewogICAgImNwdXNldCI6ICIwLTMiCiAgfQp9Cg==
              mode: 420
              overwrite: true
              path: /etc/kubernetes/openshift-workload-pinning
              user:
                name: root
    - fileName: PerformanceProfile.yaml
      policyName: "config-policy"
      metadata:
        name: openshift-worker-node-performance-profile
  spec:
    cpu: 4
      isolated: "4-47"
      reserved: "0-3"
    hugepages:
      defaultHugepagesSize: 1G
      pages:
        - size: 1G
          count: 32
    realTimeKernel:
      enabled: true
- fileName: TunedPerformancePatch.yaml
  policyName: "config-policy"
  metadata:
    name: performance-patch-worker
  spec:
    profile:
      - name: performance-patch-worker
        data: |
          [main]
          summary=Configuration changes profile inherited from performance created tuned
          include=openshift-node-performance-openshift-worker-node-performance-profile
          [bootloader]
          cmdline_crash=nohz_full=4-47 5
          [sysctl]
          kernel.timer_migration=1
          [scheduler]
          group.ice-ptp=0:f:10:*:ice-ptp.*
          [service]
          service.stalld=start,enable
          service.chronyd=stop,disable
    recommend:
    - profile: performance-patch-worker
```

1. The policies are applied to all clusters with this label.

2. The **MCP** field must be set to **worker**.

3. This generic **MachineConfig** CR is used to configure workload partitioning on the worker node.

4. The **cpu.isolated** and **cpu.reserved** fields must be configured for each particular hardware platform.

5. The **cmdline_crash** CPU set must match the **cpu.isolated** set in the **PerformanceProfile** section.

A generic **MachineConfig** CR is used to configure workload partitioning on the worker node. You can generate the content of **crio** and **kubelet** configuration files.

2. Add the created policy template to the Git repository monitored by the ArgoCD **policies** application.

3. Add the policy in the **kustomization.yaml** file.

4. Commit the changes in Git, and then push to the Git repository being monitored by the GitOps ZTP ArgoCD application.

5. To remediate the new policies to your spoke cluster, create a TALM custom resource:

```
$ cat <<EOF | oc apply -f -
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: example-sno-worker-policies
  namespace: default
spec:
  backup: false
  clusters:
  - example-sno
  enable: true
  managedPolicies:
  - group-du-sno-config-policy
  - example-sno-workers-config-policy
  - example-sno-config-policy
  preCaching: false
  remediationStrategy:
    maxConcurrency: 1
EOF
```

17.13.5. Adding worker nodes to single-node OpenShift clusters with GitOps ZTP

You can add one or more worker nodes to existing single-node OpenShift clusters to increase available CPU resources in the cluster.

Prerequisites

- Install and configure RHACM 2.6 or later in an OpenShift Container Platform 4.11 or later bare-metal hub cluster
- Install Topology Aware Lifecycle Manager in the hub cluster
- Install Red Hat OpenShift GitOps in the hub cluster
- Use the GitOps ZTP **ztp-site-generate** container image version 4.12 or later
- Deploy a managed single-node OpenShift cluster with GitOps ZTP
- Configure the Central Infrastructure Management as described in the RHACM documentation
- Configure the DNS serving the cluster to resolve the internal API endpoint **api-int.<cluster_name>.<base_domain>**

Procedure

1. If you deployed your cluster by using the **example-sno.yaml SiteConfig** manifest, add your new worker node to the **spec.clusters['example-sno'].nodes** list:

```
nodes:
- hostName: "example-node2.example.com"
```

```
role: "worker"
bmcAddress: "idrac-virtualmedia+https://[1111:2222:3333:4444::bbbb:1]/redfish/v1/Systems/System.Embedded.1"

bmcCredentialsName:
  name: "example-node2-bmh-secret"
bootMACAddress: "AA:BB:CC:DD:EE:11"
bootMode: "UEFI"
nodeNetwork:
  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
            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
```

2. Create a BMC authentication secret for the new host, as referenced by the **bmcCredentialsName** field in the **spec.nodes** section of your **SiteConfig** file:

```
apiVersion: v1
data:
  password: "password"
  username: "username"
kind: Secret
metadata:
  name: "example-node2-bmh-secret"
  namespace: example-sno
type: Opaque
```

3. Commit the changes in Git, and then push to the Git repository that is being monitored by the GitOps ZTP ArgoCD application.
   When the ArgoCD **cluster** application synchronizes, two new manifests appear on the hub cluster generated by the ZTP plugin:

- **BareMetalHost**
- **NMStateConfig**

IMPORTANT

The **cpuset** field should not be configured for the worker node. Workload partitioning for worker nodes is added through management policies after the node installation is complete.

Verification

You can monitor the installation process in several ways.

- Check if the preprovisioning images are created by running the following command:

```
$ oc get ppimg -n example-sno
```

Example output

```
NAMESPACE      NAME           READY   REASON
example-sno    example-sno    True    ImageCreated
example-sno    example-node2  True    ImageCreated
```

- Check the state of the bare-metal hosts:

```
$ oc get bmh -n example-sno
```

Example output

```
NAME            STATE          CONSUMER   ONLINE   ERROR   AGE
example-sno     provisioned               true             69m
example-node2   provisioning              true             4m50s 1
```

1 The **provisioning** state indicates that node booting from the installation media is in progress.

- Continuously monitor the installation process:

a. Watch the agent install process by running the following command:

```
$ oc get agent -n example-sno --watch
```

Example output

```
NAME                                   CLUSTER       APPROVED   ROLE     STAGE
671bc05d-5358-8940-ec12-d9ad22804faa   example-sno   true       master   Done
[...]
14fd821b-a35d-9cba-7978-00ddf535ff37   example-sno   true       worker   Starting
installation
14fd821b-a35d-9cba-7978-00ddf535ff37   example-sno   true       worker   Installing
14fd821b-a35d-9cba-7978-00ddf535ff37   example-sno   true       worker   Writing image
```

```
to disk
[...]
14fd821b-a35d-9cba-7978-00ddf535ff37   example-sno   true      worker   Waiting for
control plane
[...]
14fd821b-a35d-9cba-7978-00ddf535ff37   example-sno   true      worker   Rebooting
14fd821b-a35d-9cba-7978-00ddf535ff37   example-sno   true      worker   Done
```

b. When the worker node installation is finished, the worker node certificates are approved automatically. At this point, the worker appears in the **ManagedClusterInfo** status. Run the following command to see the status:

```
$ oc get managedclusterinfo/example-sno -n example-sno -o \
jsonpath='{range .status.nodeList[*]}{.name}{"\t"}{.conditions}{"\t"}{.labels}{"\n"}{end}'
```

Example output

```
example-sno [{"status":"True","type":"Ready"}] {"node-
role.kubernetes.io/master":"","node-role.kubernetes.io/worker":""}
example-node2 [{"status":"True","type":"Ready"}] {"node-role.kubernetes.io/worker":""}
```

17.14. PRE-CACHING IMAGES FOR SINGLE-NODE OPENSHIFT DEPLOYMENTS

In environments with limited bandwidth where you use the GitOps zero touch provisioning (ZTP) solution to deploy a large number of clusters, you want to avoid downloading all the images that are required for bootstrapping and installing OpenShift Container Platform. The limited bandwidth at remote single-node OpenShift sites can cause long deployment times. The factory-precaching-cli tool allows you to pre-stage servers before shipping them to the remote site for ZTP provisioning.

The factory-precaching-cli tool does the following:

- Downloads the RHCOS rootfs image that is required by the minimal ISO to boot.
- Creates a partition from the installation disk labelled as **data**.
- Formats the disk in xfs.
- Creates a GUID Partition Table (GPT) data partition at the end of the disk, where the size of the partition is configurable by the tool.
- Copies the container images required to install OpenShift Container Platform.
- Copies the container images required by ZTP to install OpenShift Container Platform.
- Optional: Copies Day-2 Operators to the partition.

IMPORTANT

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

For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

17.14.1. Getting the factory-precaching-cli tool

The factory-precaching-cli tool Go binary is publicly available in the Telco RAN tools container image. The factory-precaching-cli tool Go binary in the container image is executed on the server running an RHCOS live image using **podman**. If you are working in a disconnected environment or have a private registry, you need to copy the image there so you can download the image to the server.

Procedure

- Pull the factory-precaching-cli tool image by running the following command:

```
# podman pull quay.io/openshift-kni/telco-ran-tools:latest
```

Verification

- To check that the tool is available, query the current version of the factory-precaching-cli tool Go binary:

```
# podman run quay.io/openshift-kni/telco-ran-tools:latest -- factory-precaching-cli -v
```

Example output

```
factory-precaching-cli version 20221018.120852+main.feecf17
```

17.14.2. Booting from a live operating system image

You can use the factory-precaching-cli tool with to boot servers where only one disk is available and external disk drive cannot be attached to the server.

WARNING

RHCOS requires the disk to not be in use when the disk is about to be written with an RHCOS image.

Depending on the server hardware, you can mount the RHCOS live ISO on the blank server using one of the following methods:

- Using the Dell RACADM tool on a Dell server.
- Using the HPONCFG tool on a HP server.
- Using the Redfish BMC API.

NOTE

It is recommended to automate the mounting procedure. To automate the procedure, you need to pull the required images and host them on a local HTTP server.

Prerequisites

- You powered up the host.
- You have network connectivity to the host.

PROCEDURE

This example procedure uses the Redfish BMC API to mount the RHCOS live ISO.

1. Mount the RHCOS live ISO:

   a. Check virtual media status:

   ```
   $ curl --globoff -H "Content-Type: application/json" -H \
   "Accept: application/json" -k -X GET --user ${username_password} \
   https://$BMC_ADDRESS/redfish/v1/Managers/Self/VirtualMedia/1 | python -m json.tool
   ```

   b. Mount the ISO file as a virtual media:

   ```
   $ curl --globoff -L -w "%{http_code} %{url_effective}\\n" -ku ${username_password} -H
   "Content-Type: application/json" -H "Accept: application/json" -d '{"Image":
   "http://[$HTTPd_IP]/RHCOS-live.iso"}' -X POST
   https://$BMC_ADDRESS/redfish/v1/Managers/Self/VirtualMedia/1/Actions/VirtualMedia.Ins
   ertMedia
   ```

   c. Set the boot order to boot from the virtual media once:

   ```
   $ curl --globoff  -L -w "%{http_code} %{url_effective}\\n"  -ku ${username_password}  -H
   "Content-Type: application/json" -H "Accept: application/json" -d '{"Boot":{
   "BootSourceOverrideEnabled": "Once", "BootSourceOverrideTarget": "Cd",
   "BootSourceOverrideMode": "UEFI"}}' -X PATCH
   https://$BMC_ADDRESS/redfish/v1/Systems/Self
   ```

2. Reboot and ensure that the server is booting from virtual media.

Additional resources

- For more information about the **butane** utility, see About Butane.
- For more information about creating a custom live RHCOS ISO, see Creating a custom live RHCOS ISO for remote server access.

- For more information about using the Dell RACADM tool, see Integrated Dell Remote Access Controller 9 RACADM CLI Guide.
- For more information about using the HP HPONCFG tool, see Using HPONCFG.
- For more information about using the Redfish BMC API, see Booting from an HTTP-hosted ISO image using the Redfish API.

17.14.3. Partitioning the disk

To run the full pre-caching process, you have to boot from a live ISO and use the factory-precaching-cli tool from a container image to partition and pre-cache all the artifacts required.

A live ISO or RHCOS live ISO is required because the disk must not be in use when the operating system (RHCOS) is written to the device during the provisioning. Single-disk servers can also be enabled with this procedure.

Prerequisites

- You have a disk that is not partitioned.
- You have access to the **quay.io/openshift-kni/telco-ran-tools:latest** image.
- You have enough storage to install OpenShift Container Platform and pre-cache the required images.

Procedure

1. Verify that the disk is cleared:

```
# lsblk
```

Example output

```
NAME    MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
loop0     7:0    0  93.8G  0 loop /run/ephemeral
loop1     7:1    0 897.3M  1 loop /sysroot
sr0      11:0    1   999M  0 rom  /run/media/iso
nvme0n1 259:1    0   1.5T  0 disk
```

2. Erase any file system, RAID or partition table signatures from the device:

```
# wipefs -a /dev/nvme0n1
```

Example output

```
/dev/nvme0n1: 8 bytes were erased at offset 0x00000200 (gpt): 45 46 49 20 50 41 52 54
/dev/nvme0n1: 8 bytes were erased at offset 0x1749a955e00 (gpt): 45 46 49 20 50 41 52 54
/dev/nvme0n1: 2 bytes were erased at offset 0x000001fe (PMBR): 55 aa
```

IMPORTANT

The tool fails if the disk is not empty because it uses partition number 1 of the device for pre-caching the artifacts.

17.14.3.1. Creating the partition

Once the device is ready, you create a single partition and a GPT partition table. The partition is automatically labelled as **data** and created at the end of the device. Otherwise, the partition will be overridden by the **coreos-installer**.

IMPORTANT

The **coreos-installer** requires the partition to be created at the end of the device and to be labelled as **data**. Both requirements are necessary to save the partition when writing the RHCOS image to the disk.

Prerequisites

- The container must run as **privileged** due to formatting host devices.
- You have to mount the /**dev** folder so that the process can be executed inside the container.

Procedure

In the following example, the size of the partition is 250 GiB due to allow pre-caching the DU profile for Day 2 Operators.

1. Run the container as **privileged** and partition the disk:

```
# podman run -v /dev:/dev --privileged \
--rm quay.io/openshift-kni/telco-ran-tools:latest -- \
factory-precaching-cli partition \ ①
-d /dev/nvme0n1 \ ②
-s 250 ③
```

① Specifies the partitioning function of the factory-precaching-cli tool.

② Defines the root directory on the disk.

③ Defines the size of the disk in GB.

2. Check the storage information:

```
# lsblk
```

Example output

```
NAME        MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
loop0         7:0    0  93.8G  0 loop /run/ephemeral
loop1         7:1    0 897.3M  1 loop /sysroot
sr0          11:0    1   999M  0 rom  /run/media/iso
nvme0n1     259:1    0   1.5T  0 disk
└─nvme0n1p1 259:3    0   250G  0 part
```

–

Verification

You must verify that the following requirements are met:

- The device has a GPT partition table
- The partition uses the latest sectors of the device.
- The partition is correctly labeled as **data**.

Query the disk status to verify that the disk is partitioned as expected:

```
# gdisk -l /dev/nvme0n1
```

Example output

```
GPT fdisk (gdisk) version 1.0.3

Partition table scan:
  MBR: protective
  BSD: not present
  APM: not present
  GPT: present

Found valid GPT with protective MBR; using GPT.
Disk /dev/nvme0n1: 3125627568 sectors, 1.5 TiB
Model: Dell Express Flash PM1725b 1.6TB SFF
Sector size (logical/physical): 512/512 bytes
Disk identifier (GUID): CB5A9D44-9B3C-4174-A5C1-C64957910B61
Partition table holds up to 128 entries
Main partition table begins at sector 2 and ends at sector 33
First usable sector is 34, last usable sector is 3125627534
Partitions will be aligned on 2048-sector boundaries
Total free space is 2601338846 sectors (1.2 TiB)

Number  Start (sector)    End (sector)  Size       Code  Name
   1      2601338880      3125627534   250.0 GiB   8300  data
```

17.14.3.2. Mounting the partition

After verifying that the disk is partitioned correctly, you can mount the device into **/mnt**.

IMPORTANT

It is recommended to mount the device into /**mnt** because that mounting point is used during ZTP preparation.

1. Verify that the partition is formatted as **xfs**:

```
# lsblk -f /dev/nvme0n1
```

Example output

```
NAME        FSTYPE LABEL UUID                                 MOUNTPOINT
nvme0n1
└─nvme0n1p1 xfs          1bee8ea4-d6cf-4339-b690-a76594794071
```

2. Mount the partition:

```
# mount /dev/nvme0n1p1 /mnt/
```

Verification

- Check that the partition is mounted:

```
# lsblk
```

Example output

```
NAME        MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
loop0         7:0    0  93.8G  0 loop /run/ephemeral
loop1         7:1    0 897.3M  1 loop /sysroot
sr0          11:0    1   999M  0 rom  /run/media/iso
nvme0n1     259:1    0   1.5T  0 disk
└─nvme0n1p1 259:2    0   250G  0 part /var/mnt 1
```

1 The mount point is **/var/mnt** because the **/mnt** folder in RHCOS is a link to **/var/mnt**.

17.14.4. Downloading the images

The factory-precaching-cli tool allows you to download the following images to your partitioned server:

- OpenShift Container Platform images
- Operator images that are included in the distributed unit (DU) profile for 5G RAN sites
- Operator images from disconnected registries

NOTE

The list of available Operator images can vary in different OpenShift Container Platform releases.

17.14.4.1. Downloading with parallel workers

The factory-precaching-cli tool uses parallel workers to download multiple images simultaneously. You can configure the number of workers with the **--parallel** or **-p** option. The default number is set to 80% of the available CPUs to the server.

NOTE

Your login shell may be restricted to a subset of CPUs, which reduces the CPUs available to the container. To remove this restriction, you can precede your commands with **taskset 0xffffffff**, for example:

```
# taskset 0xffffffff podman run --rm quay.io/openshift-kni/telco-ran-tools:latest factory-
precaching-cli download --help
```

17.14.4.2. Preparing to download the OpenShift Container Platform images

To download OpenShift Container Platform container images, you need to know the multicluster engine (MCE) version. When you use the **--du-profile** flag, you also need to specify the Red Hat Advanced Cluster Management (RHACM) version running in the hub cluster that is going to provision the single-node OpenShift.

Prerequisites

- You have RHACM and MCE installed.
- You partitioned the storage device.
- You have enough space for the images on the partitioned device.
- You connected the bare-metal server to the Internet.
- You have a valid pull secret.

Procedure

1. Check the RHACM and MCE version by running the following commands in the hub cluster:

```
$ oc get csv -A | grep -i advanced-cluster-management
```

Example output

```
open-cluster-management                          advanced-cluster-management.v2.6.3
Advanced Cluster Management for Kubernetes   2.6.3                advanced-cluster-
management.v2.6.3                Succeeded
```

```
$ oc get csv -A | grep -i multicluster-engine
```

Example output

```
multicluster-engine                              cluster-group-upgrades-operator.v0.0.3       cluster-
group-upgrades-operator              0.0.3                                                    Pending
multicluster-engine                              multicluster-engine.v2.1.4                multicluster
engine for Kubernetes           2.1.4                multicluster-engine.v2.0.3
Succeeded
multicluster-engine                              openshift-gitops-operator.v1.5.7            Red Hat
OpenShift GitOps                  1.5.7                openshift-gitops-operator.v1.5.6-
0.1664915551.p   Succeeded
```

```
multicluster-engine                         openshift-pipelines-operator-rh.v1.6.4       Red Hat
OpenShift Pipelines             1.6.4              openshift-pipelines-operator-rh.v1.6.3
Succeeded
```

2. To access the container registry, copy a valid pull secret on the server to be installed:

   a. Create the **.docker** folder:

   ```
   $ mkdir /root/.docker
   ```

   b. Copy the valid pull in the **config.json** file to the previously created **.docker/** folder:

   ```
   $ cp config.json /root/.docker/config.json ①
   ```

   ① **/root/.docker/config.json** is the default path where **podman** checks for the login credentials for the registry.

NOTE

If you use a different registry to pull the required artifacts, you need to copy the proper pull secret. If the local registry uses TLS, you need to include the certificates from the registry as well.

17.14.4.3. Downloading the OpenShift Container Platform images

The factory-precaching-cli tool allows you to pre-cache all the container images required to provision a specific OpenShift Container Platform release.

Procedure

- Pre-cache the release by running the following command:

  ```
  # podman run -v /mnt:/mnt -v /root/.docker:/root/.docker --privileged --rm quay.io/openshift-
  kni/telco-ran-tools -- \
    factory-precaching-cli download \ ①
    -r 4.12.0 \ ②
    --acm-version 2.6.3 \ ③
    --mce-version 2.1.4 \ ④
    -f /mnt \ ⑤
    --img quay.io/custom/repository ⑥
  ```

  ① Specifies the downloading function of the factory-precaching-cli tool.

  ② Defines the OpenShift Container Platform release version.

  ③ Defines the RHACM version.

  ④ Defines the MCE version.

  ⑤ Defines the folder where you want to download the images on the disk.

  ⑥ Optional. Defines the repository where you store your additional images. These images are downloaded and pre-cached on the disk.

Example output

```
Generated /mnt/imageset.yaml
Generating list of pre-cached artifacts...
Processing artifact [1/176]: ocp-v4.0-art-
dev@sha256_6ac2b96bf4899c01a87366fd0feae9f57b1b61878e3b5823da0c3f34f707fbf5
Processing artifact [2/176]: ocp-v4.0-art-
dev@sha256_f48b68d5960ba903a0d018a10544ae08db5802e21c2fa5615a14fc58b1c1657c
Processing artifact [3/176]: ocp-v4.0-art-
dev@sha256_a480390e91b1c07e10091c3da2257180654f6b2a735a4ad4c3b69dbdb77bbc06

Processing artifact [4/176]: ocp-v4.0-art-
dev@sha256_ecc5d8dbd77e326dba6594ff8c2d091eefbc4d90c963a9a85b0b2f0e6155f995
Processing artifact [5/176]: ocp-v4.0-art-
dev@sha256_274b6d561558a2f54db08ea96df9892315bb773fc203b1dbcea418d20f4c7ad1
Processing artifact [6/176]: ocp-v4.0-art-
dev@sha256_e142bf5020f5ca0d1bdda0026bf97f89b72d21a97c9cc2dc71bf85050e822bbf
...
Processing artifact [175/176]: ocp-v4.0-art-
dev@sha256_16cd7eda26f0fb0fc965a589e1e96ff8577e560fcd14f06b5fda1643036ed6c8
Processing artifact [176/176]: ocp-v4.0-art-
dev@sha256_cf4d862b4a4170d4f611b39d06c31c97658e309724f9788e155999ae51e7188f
...
Summary:

Release:                4.12.0
Hub Version:              2.6.3
ACM Version:              2.6.3
MCE Version:              2.1.4
Include DU Profile:        No
Workers:                83
```

Verification

- Check that all the images are compressed in the target folder of server:

```
$ ls -l /mnt 1
```

1 It is recommended that you pre-cache the images in the **/mnt** folder.

Example output

```
-rw-r--r--. 1 root root  136352323 Oct 31 15:19 ocp-v4.0-art-
dev@sha256_edec37e7cd8b1611d0031d45e7958361c65e2005f145b471a8108f1b54316c07.t
gz
-rw-r--r--. 1 root root  156092894 Oct 31 15:33 ocp-v4.0-art-
dev@sha256_ee51b062b9c3c9f4fe77bd5b3cc9a3b12355d040119a1434425a824f137c61a9.tg
z
-rw-r--r--. 1 root root  172297800 Oct 31 15:29 ocp-v4.0-art-
dev@sha256_ef23d9057c367a36e4a5c4877d23ee097a731e1186ed28a26c8d21501cd82718.t
gz
-rw-r--r--. 1 root root  171539614 Oct 31 15:23 ocp-v4.0-art-
dev@sha256_f0497bb63ef6834a619d4208be9da459510df697596b891c0c633da144dbb025.t
gz
```

```
-rw-r--r--. 1 root root  160399150 Oct 31 15:20 ocp-v4.0-art-
dev@sha256_f0c339da117cde44c9aae8d0bd054bceb6f19fdb191928f6912a703182330ac2.tgz

-rw-r--r--. 1 root root  175962005 Oct 31 15:17 ocp-v4.0-art-
dev@sha256_f19dd2e80fb41ef31d62bb8c08b339c50d193fdb10fc39cc15b353cbbfeb9b24.tgz

-rw-r--r--. 1 root root  174942008 Oct 31 15:33 ocp-v4.0-art-
dev@sha256_f1dbb81fa1aa724e96dd2b296b855ff52a565fbef003d08030d63590ae6454df.tgz

-rw-r--r--. 1 root root  246693315 Oct 31 15:31 ocp-v4.0-art-
dev@sha256_f44dcf2c94e4fd843cbbf9b11128df2ba856cd813786e42e3da1fdfb0f6ddd01.tgz
-rw-r--r--. 1 root root  170148293 Oct 31 15:00 ocp-v4.0-art-
dev@sha256_f48b68d5960ba903a0d018a10544ae08db5802e21c2fa5615a14fc58b1c1657c.tg
z
-rw-r--r--. 1 root root  168899617 Oct 31 15:16 ocp-v4.0-art-
dev@sha256_f5099b0989120a8d08a963601214b5c5cb23417a707a8624b7eb52ab788a7f75.t
gz
-rw-r--r--. 1 root root  176592362 Oct 31 15:05 ocp-v4.0-art-
dev@sha256_f68c0e6f5e17b0b0f7ab2d4c39559ea89f900751e64b97cb42311a478338d9c3.tg
z
-rw-r--r--. 1 root root  157937478 Oct 31 15:37 ocp-v4.0-art-
dev@sha256_f7ba33a6a9db9cfc4b0ab0f368569e19b9fa08f4c01a0d5f6a243d61ab781bd8.tgz

-rw-r--r--. 1 root root  145535253 Oct 31 15:26 ocp-v4.0-art-
dev@sha256_f8f098911d670287826e9499806553f7a1dd3e2b5332abbec740008c36e84de5.t
gz
-rw-r--r--. 1 root root  158048761 Oct 31 15:40 ocp-v4.0-art-
dev@sha256_f914228ddbb99120986262168a705903a9f49724ffa958bb4bf12b2ec1d7fb47.tgz

-rw-r--r--. 1 root root  167914526 Oct 31 15:37 ocp-v4.0-art-
dev@sha256_fa3ca9401c7a9efda0502240aeb8d3ae2d239d38890454f17fe5158b62305010.tg
z
-rw-r--r--. 1 root root  164432422 Oct 31 15:24 ocp-v4.0-art-
dev@sha256_fc4783b446c70df30b3120685254b40ce13ba6a2b0bf8fb1645f116cf6a392f1.tgz

-rw-r--r--. 1 root root  306643814 Oct 31 15:11
troubleshoot@sha256_b86b8aea29a818a9c22944fd18243fa0347c7a2bf1ad8864113ff2bb2d8
e0726.tgz
```

17.14.4.4. Downloading the Operator images

You can also pre-cache Day-2 Operators used in the 5G Radio Access Network (RAN) Distributed Unit (DU) cluster configuration. The Day-2 Operators depend on the installed OpenShift Container Platform version.

IMPORTANT

You need to include the RHACM hub and MCE Operator versions by using the **--acm-version** and **--mce-version** flags so the factory-precaching-cli tool can pre-cache the appropriate containers images for the RHACM and MCE Operators.

Procedure

- Pre-cache the Operator images:

```
# podman run -v /mnt:/mnt -v /root/.docker:/root/.docker --privileged --rm quay.io/openshift-
kni/telco-ran-tools:latest -- factory-precaching-cli download \ 1
  -r 4.12.0 \ 2
  --acm-version 2.6.3 \ 3
  --mce-version 2.1.4 \ 4
  -f /mnt \ 5
  --img quay.io/custom/repository 6
  --du-profile -s 7
```

1 Specifies the downloading function of the factory-precaching-cli tool.

2 Defines the OpenShift Container Platform release version.

3 Defines the RHACM version.

4 Defines the MCE version.

5 Defines the folder where you want to download the images on the disk.

6 Optional. Defines the repository where you store your additional images. These images are downloaded and pre-cached on the disk.

7 Specifies pre-caching the Operators included in the DU configuration.

Example output

```
Generated /mnt/imageset.yaml
Generating list of pre-cached artifacts...
Processing artifact [1/379]: ocp-v4.0-art-
dev@sha256_7753a8d9dd5974be8c90649aadd7c914a3d8a1f1e016774c7ac7c9422e9f9958
Processing artifact [2/379]: ose-kube-rbac-
proxy@sha256_c27a7c01e5968aff16b6bb6670423f992d1a1de1a16e7e260d12908d3322431c

Processing artifact [3/379]: ocp-v4.0-art-
dev@sha256_370e47a14c798ca3f8707a38b28cfc28114f492bb35fe1112e55d1eb51022c99
...
Processing artifact [378/379]: ose-local-storage-
operator@sha256_0c81c2b79f79307305e51ce9d3837657cf9ba5866194e464b4d1b299f85034
d0
Processing artifact [379/379]: multicluster-operators-channel-
rhel8@sha256_c10f6bbb84fe36e05816e873a72188018856ad6aac6cc16271a1b3966f73ceb3


...
Summary:

Release:                4.12.0
Hub Version:              2.6.3
ACM Version:              2.6.3
MCE Version:              2.1.4
Include DU Profile:         Yes
Workers:                83
```

17.14.4.5. Pre-caching custom images in disconnected environments

The **--generate-imageset** argument stops the factory-precaching-cli tool after the **ImageSetConfiguration** custom resource (CR) is generated. This allows you to customize the **ImageSetConfiguration** CR before downloading any images. After you customized the CR, you can use the **--skip-imageset** argument to download the images that you specified in the **ImageSetConfiguration** CR.

You can customize the **ImageSetConfiguration** CR in the following ways:

- Add Operators and additional images
- Remove Operators and additional images
- Change Operator and catalog sources to local or disconnected registries

Procedure

1. Pre-cache the images:

```
# podman run -v /mnt:/mnt -v /root/.docker:/root/.docker --privileged --rm quay.io/openshift-
kni/telco-ran-tools:latest -- factory-precaching-cli download \ ❶
  -r 4.12.0 \ ❷
  --acm-version 2.6.3 \ ❸
  --mce-version 2.1.4 \ ❹
  -f /mnt \ ❺
  --img quay.io/custom/repository ❻
  --du-profile -s \ ❼
  --generate-imageset ❽
```

❶ Specifies the downloading function of the factory-precaching-cli tool.

❷ Defines the OpenShift Container Platform release version.

❸ Defines the RHACM version.

❹ Defines the MCE version.

❺ Defines the folder where you want to download the images on the disk.

❻ Optional. Defines the repository where you store your additional images. These images are downloaded and pre-cached on the disk.

❼ Specifies pre-caching the Operators included in the DU configuration.

❽ The **--generate-imageset** argument generates the **ImageSetConfiguration** CR only, which allows you to customize the CR.

Example output

```
Generated /mnt/imageset.yaml
```

Example ImageSetConfiguration CR

```
apiVersion: mirror.openshift.io/v1alpha2
```

```
kind: ImageSetConfiguration
mirror:
  platform:
    channels:
    - name: stable-4.12
      minVersion: 4.12.0 1
      maxVersion: 4.12.0
  additionalImages:
    - name: quay.io/custom/repository
  operators:
    - catalog: registry.redhat.io/redhat/redhat-operator-index:v4.12
      packages:
        - name: advanced-cluster-management 2
          channels:
             - name: 'release-2.6'
               minVersion: 2.6.3
               maxVersion: 2.6.3
        - name: multicluster-engine 3
          channels:
             - name: 'stable-2.1'
               minVersion: 2.1.4
               maxVersion: 2.1.4
        - name: local-storage-operator 4
          channels:
            - name: 'stable'
        - name: ptp-operator 5
          channels:
            - name: 'stable'
        - name: sriov-network-operator 6
          channels:
            - name: 'stable'
        - name: cluster-logging 7
          channels:
            - name: 'stable'
        - name: lvms-operator 8
          channels:
            - name: 'stable-4.12'
        - name: amq7-interconnect-operator 9
          channels:
            - name: '1.10.x'
        - name: bare-metal-event-relay 10
          channels:
            - name: 'stable'
    - catalog: registry.redhat.io/redhat/certified-operator-index:v4.12
      packages:
        - name: sriov-fec 11
          channels:
            - name: 'stable'
```

1 The platform versions match the versions passed to the tool.

2 3 The versions of RHACM and MCE Operators match the versions passed to the tool.

4 5 6 7 8 9 10 11 The CR contains all the specified DU Operators.

2. Customize the catalog resource in the CR:

   ```
   apiVersion: mirror.openshift.io/v1alpha2
   kind: ImageSetConfiguration
   mirror:
     platform:
   [...]
     operators:
       - catalog: eko4.cloud.lab.eng.bos.redhat.com:8443/redhat/certified-operator-index:v4.12
         packages:
           - name: sriov-fec
             channels:
               - name: 'stable'
   ```

   When you download images by using a local or disconnected registry, you have to first add certificates for the registries that you want to pull the content from.

3. To avoid any errors, copy the registry certificate into your server:

   ```
   # cp /tmp/eko4-ca.crt /etc/pki/ca-trust/source/anchors/.
   ```

4. Then, update the certificates trust store:

   ```
   # update-ca-trust
   ```

5. Mount the host **/etc/pki** folder into the factory-cli image:

   ```
   # podman run -v /mnt:/mnt -v /root/.docker:/root/.docker -v /etc/pki:/etc/pki --privileged --rm
   quay.io/openshift-kni/telco-ran-tools:latest -- \
   factory-precaching-cli download \ ①
      -r 4.12.0 \ ②
      --acm-version 2.6.3 \ ③
      --mce-version 2.1.4 \ ④
      -f /mnt \ ⑤
      --img quay.io/custom/repository ⑥
      --du-profile -s \ ⑦
      --skip-imageset ⑧
   ```

   ① Specifies the downloading function of the factory-precaching-cli tool.

   ② Defines the OpenShift Container Platform release version.

   ③ Defines the RHACM version.

   ④ Defines the MCE version.

   ⑤ Defines the folder where you want to download the images on the disk.

   ⑥ Optional. Defines the repository where you store your additional images. These images are downloaded and pre-cached on the disk.

   ⑦ Specifies pre-caching the Operators included in the DU configuration.

   ⑧

The **--skip-imageset** argument allows you to download the images that you specified in your customized **ImageSetConfiguration** CR.

6. Download the images without generating a new **imageSetConfiguration** CR:

```
# podman run -v /mnt:/mnt -v /root/.docker:/root/.docker --privileged --rm quay.io/openshift-
kni/telco-ran-tools:latest -- factory-precaching-cli download -r 4.12.0 \
--acm-version 2.6.3 --mce-version 2.1.4 -f /mnt \
--img quay.io/custom/repository \
--du-profile -s \
--skip-imageset
```

Additional resources

- To access the online Red Hat registries, see OpenShift installation customization tools.
- For more information about using the multicluster engine, see About cluster lifecycle with the multicluster engine operator.

17.14.5. Pre-caching images in ZTP

The **SiteConfig** manifest defines how an OpenShift cluster is to be installed and configured. In the ZTP provisioning workflow, the factory-precaching-cli tool requires the following additional fields in the **SiteConfig** manifest:

- **clusters.ignitionConfigOverride**
- **nodes.installerArgs**
- **nodes.ignitionConfigOverride**

Example SiteConfig with additional fields

```
apiVersion: ran.openshift.io/v1
kind: SiteConfig
metadata:
  name: "example-5g-lab"
  namespace: "example-5g-lab"
spec:
  baseDomain: "example.domain.redhat.com"
  pullSecretRef:
    name: "assisted-deployment-pull-secret"
  clusterImageSetNameRef: "img4.9.10-x86-64-appsub"
  sshPublicKey: "ssh-rsa ..."
  clusters:
  - clusterName: "sno-worker-0"
    clusterLabels:
      group-du-sno: ""
      common-411: true
      sites : "example-5g-lab"
      vendor: "OpenShift"
    clusterNetwork:
      - cidr: 10.128.0.0/14
        hostPrefix: 23
  machineNetwork:
   - cidr: 10.19.32.192/26
  serviceNetwork:
   - 172.30.0.0/16
  networkType: "OVNKubernetes"
  additionalNTPSources:
   - clock.corp.redhat.com
  ignitionConfigOverride: '{"ignition":{"version":"3.1.0"},"systemd":{"units":[{"name":"var-
mnt.mount","enabled":true,"contents":"[Unit]\nDescription=Mount partition with
artifacts\nBefore=precache-images.service\nBindsTo=precache-
images.service\nStopWhenUnneeded=true\n\n[Mount]\nWhat=/dev/disk/by-
partlabel/data\nWhere=/var/mnt\nType=xfs\nTimeoutSec=30\n\n[Install]\nRequiredBy=precache-
images.service"},{"name":"precache-images.service","enabled":true,"contents":"
[Unit]\nDescription=Extracts the precached images in discovery stage\nAfter=var-
mnt.mount\nBefore=agent.service\n\n[Service]\nType=oneshot\nUser=root\nWorkingDirectory=/var/mnt\
nExecStart=bash /usr/local/bin/extract-ai.sh\n#TimeoutStopSec=30\n\n[Install]\nWantedBy=multi-
user.target default.target\nWantedBy=agent.service"}]},"storage":{"files":
[{"overwrite":true,"path":"/usr/local/bin/extract-ai.sh","mode":755,"user":{"name":"root"},"contents":
{"source":"data:,%23%21%2Fbin%2Fbash%0A%0AFOLDER%3D%22%24%7BFOLDER%3A-
%24%28pwd%29%7D%22%0AOCP_RELEASE_LIST%3D%22%24%7BOCP_RELEASE_LIST%3A-
ai-
images.txt%7D%22%0ABINARY_FOLDER%3D%2Fvar%2Fmnt%0A%0Apushd%20%24FOLDER%0
A%0Atotal_copies%3D%24%28sort%20-
u%20%24BINARY_FOLDER%2F%24OCP_RELEASE_LIST%20%7C%20wc%20-
l%29%20%20%23%20Required%20to%20keep%20track%20of%20the%20pull%20task%20vs%20tot
al%0Acurrent_copy%3D1%0A%0Awhile%20read%20-
r%20line%3B%0Ado%0A%20%20uri%3D%24%28echo%20%22%24line%22%20%7C%20awk%20%
27%7Bprint%241%7D%27%29%0A%20%20%23tar%3D%24%28echo%20%22%24line%22%20%7
C%20awk%20%27%7Bprint%242%7D%27%29%0A%20%20podman%20image%20exists%20%24ur
i%0A%20%20if%20%5B%5B%20%24%3F%20-
eq%200%20%5D%5D%3B%20then%0A%20%20%20%20%20%20echo%20%22Skipping%20existin
g%20image%20%24tar%22%0A%20%20%20%20%20%20echo%20%22Copying%20%24%7Buri%7
D%20%5B%24%7Bcurrent_copy%7D%2F%24%7Btotal_copies%7D%5D%22%0A%20%20%20%20
%20%20current_copy%3D%24%28%28current_copy%20%2B%201%29%29%0A%20%20%20%20
%20%20continue%0A%20%20fi%0A%20%20tar%3D%24%28echo%20%22%24uri%22%20%7C%2
0%20rev%20%7C%20cut%20-d%20%22%2F%22%20-
f1%20%7C%20rev%20%7C%20tr%20%22%3A%22%20%22_%22%29%0A%20%20tar%20zxvf%20
%24%7Btar%7D.tgz%0A%20%20if%20%5B%20%24%3F%20-
eq%200%20%5D%3B%20then%20rm%20-
f%20%24%7Btar%7D.gz%3B%20fi%0A%20%20echo%20%22Copying%20%24%7Buri%7D%20%5B
%24%7Bcurrent_copy%7D%2F%24%7Btotal_copies%7D%5D%22%0A%20%20skopeo%20copy%20
dir%3A%2F%2F%24%28pwd%29%2F%24%7Btar%7D%20containers-
storage%3A%24%7Buri%7D%0A%20%20if%20%5B%20%24%3F%20-
eq%200%20%5D%3B%20then%20rm%20-
rf%20%24%7Btar%7D%3B%20current_copy%3D%24%28%28current_copy%20%2B%201%29%29
%3B%20fi%0Adone%20%3C%20%24%7BBINARY_FOLDER%7D%2F%24%7BOCP_RELEASE_LI
ST%7D%0A%0A%23%20workaround%20while%20https%3A%2F%2Fgithub.com%2Fopenshift%2Fa
ssisted-service%2Fpull%2F3546%0A%23cp%20%2Fvar%2Fmnt%2Fmodified-rhcos-4.10.3-x86_64-
metal.x86_64.raw.gz%20%2Fvar%2Ftmp%2F.%0A%0Aexit%200"}},
{"overwrite":true,"path":"/usr/local/bin/agent-fix-bz1964591","mode":755,"user":
{"name":"root"},"contents":
{"source":"data:,%23%21%2Fusr%2Fbin%2Fsh%0A%0A%23%20This%20script%20is%20a%20work
around%20for%20bugzilla%201964591%20where%20symlinks%20inside%20%2Fvar%2Flib%2Fcont
ainers%2F%20get%0A%23%20corrupted%20under%20some%20circumstances.%0A%23%0A%23%
20In%20order%20to%20let%20agent.service%20start%20correctly%20we%20are%20checking%20h
ere%20whether%20the%20requested%0A%23%20container%20image%20exists%20and%20in%20c
```

```
ase%20%22podman%20images%22%20returns%20an%20error%20we%20try%20removing%20the
%20faulty%0A%23%20image.%0A%23%0A%23%20In%20such%20a%20scenario%20agent.service
%20will%20detect%20the%20image%20is%20not%20present%20and%20pull%20it%20again.%20In
%20case%0A%23%20the%20image%20is%20present%20and%20can%20be%20detected%20correc
tly%2C%20no%20any%20action%20is%20required.%0A%0AIMAGE%3D%24%28echo%20%241%2
0%7C%20sed%20%27s%2F%3A.%2A%2F%2F%27%29%0Apodman%20image%20exists%20%24I
MAGE%20%7C%7C%20echo%20%22already%20loaded%22%20%7C%7C%20echo%20%22need
%20to%20be%20pulled%22%0A%23podman%20images%20%7C%20grep%20%24IMAGE%20%7C
%7C%20podman%20rmi%20--force%20%241%20%7C%7C%20true"}}]}}'
    nodes:
      - hostName: "snonode.sno-worker-0.example.domain.redhat.com"
        role: "master"
        bmcAddress: "idrac-virtualmedia+https://10.19.28.53/redfish/v1/Systems/System.Embedded.1"
        bmcCredentialsName:
          name: "worker0-bmh-secret"
        bootMACAddress: "e4:43:4b:bd:90:46"
        bootMode: "UEFI"
        rootDeviceHints:
          deviceName: /dev/nvme0n1
        cpuset: "0-1,40-41"
        installerArgs: '["--save-partlabel", "data"]'
        ignitionConfigOverride: '{"ignition":{"version":"3.1.0"},"systemd":{"units":[{"name":"var-
mnt.mount","enabled":true,"contents":"[Unit]\nDescription=Mount partition with
artifacts\nBefore=precache-ocp-images.service\nBindsTo=precache-ocp-
images.service\nStopWhenUnneeded=true\n\n[Mount]\nWhat=/dev/disk/by-
partlabel/data\nWhere=/var/mnt\nType=xfs\nTimeoutSec=30\n\n[Install]\nRequiredBy=precache-ocp-
images.service"},{"name":"precache-ocp-images.service","enabled":true,"contents":"
[Unit]\nDescription=Extracts the precached OCP images into containers storage\nAfter=var-
mnt.mount\nBefore=machine-config-daemon-pull.service nodeip-
configuration.service\n\n[Service]\nType=oneshot\nUser=root\nWorkingDirectory=/var/mnt\nExecStart=b
ash /usr/local/bin/extract-ocp.sh\nTimeoutStopSec=60\n\n[Install]\nWantedBy=multi-
user.target"}]},"storage":{"files":[{"overwrite":true,"path":"/usr/local/bin/extract-
ocp.sh","mode":755,"user":{"name":"root"},"contents":
{"source":"data:,%23%21%2Fbin%2Fbash%0A%0AFOLDER%3D%22%24%7BFOLDER%3A-
%24%28pwd%29%7D%22%0AOCP_RELEASE_LIST%3D%22%24%7BOCP_RELEASE_LIST%3A-
ocp-
images.txt%7D%22%0ABINARY_FOLDER%3D%2Fvar%2Fmnt%0A%0Apushd%20%24FOLDER%0
A%0Atotal_copies%3D%24%28sort%20-
u%20%24BINARY_FOLDER%2F%24OCP_RELEASE_LIST%20%7C%20wc%20-
l%29%20%20%23%20Required%20to%20keep%20track%20of%20the%20pull%20task%20vs%20tot
al%0Acurrent_copy%3D1%0A%0Awhile%20read%20-
r%20line%3B%0Ado%0A%20%20uri%3D%24%28echo%20%22%24line%22%20%7C%20awk%20%
27%7Bprint%241%7D%27%29%0A%20%20%23tar%3D%24%28echo%20%22%24line%22%20%7
C%20awk%20%27%7Bprint%242%7D%27%29%0A%20%20podman%20image%20exists%20%24ur
i%0A%20%20if%20%5B%5B%20%24%3F%20-
eq%200%20%5D%5D%3B%20then%0A%20%20%20%20%20%20echo%20%22Skipping%20existin
g%20image%20%24tar%22%0A%20%20%20%20%20%20echo%20%22Copying%20%24%7Buri%7
D%20%5B%24%7Bcurrent_copy%7D%2F%24%7Btotal_copies%7D%5D%22%0A%20%20%20%20
%20%20current_copy%3D%24%28%28current_copy%20%2B%201%29%29%0A%20%20%20%20
%20%20continue%0A%20%20fi%0A%20%20tar%3D%24%28echo%20%22%24uri%22%20%7C%2
0%20rev%20%7C%20cut%20-d%20%22%2F%22%20-
f1%20%7C%20rev%20%7C%20tr%20%22%3A%22%20%22_%22%29%0A%20%20tar%20zxvf%20
%24%7Btar%7D.tgz%0A%20%20if%20%5B%20%24%3F%20-
eq%200%20%5D%3B%20then%20rm%20-
f%20%24%7Btar%7D.gz%3B%20fi%0A%20%20echo%20%22Copying%20%24%7Buri%7D%20%5B
%24%7Bcurrent_copy%7D%2F%24%7Btotal_copies%7D%5D%22%0A%20%20skopeo%20copy%20
```

```
dir%3A%2F%2F%24%28pwd%29%2F%24%7Btar%7D%20containers-
storage%3A%24%7Buri%7D%0A%20%20if%20%5B%20%24%3F%20-
eq%200%20%20%5D%3B%20then%20rm%20-
rf%20%24%7Btar%7D%3B%20current_copy%3D%24%28%28current_copy%20%2B%201%29%29
%3B%20fi%0Adone%20%3C%20%24%7BBINARY_FOLDER%7D%2F%24%7BOCP_RELEASE_LI
ST%7D%0A%0Aexit%200"}}]}}'
    nodeNetwork:
     config:
      interfaces:
       - name: ens1f0
        type: ethernet
        state: up
        macAddress: "AA:BB:CC:11:22:33"
        ipv4:
         enabled: true
         dhcp: true
        ipv6:
         enabled: false
     interfaces:
      - name: "ens1f0"
       macAddress: "AA:BB:CC:11:22:33"
```

17.14.5.1. Understanding the clusters.ignitionConfigOverride field

The **clusters.ignitionConfigOverride** field adds a configuration in Ignition format during the ZTP discovery stage. The configuration includes **systemd** services in the ISO mounted in virtual media. This way, the scripts are part of the discovery RHCOS live ISO and they can be used to load the Assisted Installer (AI) images.

systemd services

The **systemd** services are **var-mnt.mount** and **precache-images.services**. The **precache-images.service** depends on the disk partition to be mounted in **/var/mnt** by the **var-mnt.mount** unit. The service calls a script called **extract-ai.sh**.

extract-ai.sh

The **extract-ai.sh** script extracts and loads the required images from the disk partition to the local container storage. When the script finishes successfully, you can use the images locally.

agent-fix-bz1964591

The **agent-fix-bz1964591** script is a workaround for an AI issue. To prevent AI from removing the images, which can force the **agent.service** to pull the images again from the registry, the **agent-fix-bz1964591** script checks if the requested container images exist.

17.14.5.2. Understanding the nodes.installerArgs field

The **nodes.installerArgs** field allows you to configure how the **coreos-installer** utility writes the RHCOS live ISO to disk. You need to indicate to save the disk partition labeled as **data** because the artifacts saved in the **data** partition are needed during the OpenShift Container Platform installation stage.

The extra parameters are passed directly to the **coreos-installer** utility that writes the live RHCOS to disk. On the next reboot, the operating system starts from the disk.

You can pass several options to the **coreos-installer** utility:

```
OPTIONS:
...
   -u, --image-url <URL>
         Manually specify the image URL

   -f, --image-file <path>
         Manually specify a local image file

   -i, --ignition-file <path>
         Embed an Ignition config from a file

   -I, --ignition-url <URL>
         Embed an Ignition config from a URL
...
      --save-partlabel <lx>...
         Save partitions with this label glob

      --save-partindex <id>...
         Save partitions with this number or range
...
      --insecure-ignition
         Allow Ignition URL without HTTPS or hash
```

17.14.5.3. Understanding the nodes.ignitionConfigOverride field

Similarly to **clusters.ignitionConfigOverride**, the **nodes.ignitionConfigOverride** field allows the addtion of configurations in Ignition format to the **coreos-installer** utility, but at the OpenShift Container Platform installation stage. When the RHCOS is written to disk, the extra configuration included in the ZTP discovery ISO is no longer available. During the discovery stage, the extra configuration is stored in the memory of the live OS.

NOTE

At this stage, the number of container images extracted and loaded is bigger than in the discovery stage. Depending on the OpenShift Container Platform release and whether you install the Day-2 Operators, the installation time can vary.

At the installation stage, the **var-mnt.mount** and **precache-ocp.services systemd** services are used.

precache-ocp.service

The **precache-ocp.service** depends on the disk partition to be mounted in **/var/mnt** by the **var-mnt.mount** unit. The **precache-ocp.service** service calls a script called **extract-ocp.sh**.

IMPORTANT

To extract all the images before the OpenShift Container Platform installation, you must execute **precache-ocp.service** before executing the **machine-config-daemon-pull.service** and **nodeip-configuration.service** services.

extract-ocp.sh

The **extract-ocp.sh** script extracts and loads the required images from the disk partition to the local container storage. When the script finishes successfully, you can use the images locally.

When you upload the **SiteConfig** and the optional **PolicyGenTemplates** custom resources (CRs) to the Git repo, which Argo CD is monitoring, you can start the ZTP workflow by syncing the CRs with the hub cluster.

17.14.6. Troubleshooting

17.14.6.1. Rendered catalog is invalid

When you download images by using a local or disconnected registry, you might see the **The rendered catalog is invalid** error. This means that you are missing certificates of the new registry you want to pull content from.

> NOTE
>
> The factory-precaching-cli tool image is built on a UBI RHEL image. Certificate paths and locations are the same on RHCOS.

Example error

```
Generating list of pre-cached artifacts...
error: unable to run command oc-mirror -c /mnt/imageset.yaml file:///tmp/fp-cli-3218002584/mirror --ignore-history --dry-run: Creating directory: /tmp/fp-cli-3218002584/mirror/oc-mirror-workspace/src/publish
Creating directory: /tmp/fp-cli-3218002584/mirror/oc-mirror-workspace/src/v2
Creating directory: /tmp/fp-cli-3218002584/mirror/oc-mirror-workspace/src/charts
Creating directory: /tmp/fp-cli-3218002584/mirror/oc-mirror-workspace/src/release-signatures
backend is not configured in /mnt/imageset.yaml, using stateless mode
backend is not configured in /mnt/imageset.yaml, using stateless mode
No metadata detected, creating new workspace
level=info msg=trying next host error=failed to do request: Head
"https://eko4.cloud.lab.eng.bos.redhat.com:8443/v2/redhat/redhat-operator-index/manifests/v4.11":
x509: certificate signed by unknown authority host=eko4.cloud.lab.eng.bos.redhat.com:8443

The rendered catalog is invalid.

Run "oc-mirror list operators --catalog CATALOG-NAME --package PACKAGE-NAME" for more information.

error: error rendering new refs: render reference
"eko4.cloud.lab.eng.bos.redhat.com:8443/redhat/redhat-operator-index:v4.11": error resolving name : failed to do request: Head "https://eko4.cloud.lab.eng.bos.redhat.com:8443/v2/redhat/redhat-operator-index/manifests/v4.11": x509: certificate signed by unknown authority
```

Procedure

1. Copy the registry certificate into your server:

   ```
   # cp /tmp/eko4-ca.crt /etc/pki/ca-trust/source/anchors/.
   ```

2. Update the certificates trust store:

   ```
   # update-ca-trust
   ```

3. Mount the host **/etc/pki** folder into the factory-cli image:

```
# podman run -v /mnt:/mnt -v /root/.docker:/root/.docker -v /etc/pki:/etc/pki --privileged -it --rm
quay.io/openshift-kni/telco-ran-tools:latest -- \
factory-precaching-cli download -r 4.11.5 --acm-version 2.5.4 \
  --mce-version 2.0.4 -f /mnt \--img quay.io/custom/repository
  --du-profile -s --skip-imageset
```