



# OpenShift Container Platform 4.7

## Scalability and performance

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



# OpenShift Container Platform 4.7 Scalability and performance

---

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

## Legal Notice

Copyright © 2021 Red Hat, Inc.

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

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

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

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

## Abstract

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

## Table of Contents

|   |           |
|---|-----------|
| <b>CHAPTER 1. RECOMMENDED PRACTICES FOR INSTALLING LARGE CLUSTERS</b> ..... | <b>6</b>  |
| 1.1. RECOMMENDED PRACTICES FOR INSTALLING LARGE SCALE CLUSTERS              | 6         |
| <b>CHAPTER 2. RECOMMENDED HOST PRACTICES</b> .....                          | <b>7</b>  |
| 2.1. RECOMMENDED NODE HOST PRACTICES  | 7         |
| 2.2. CREATING A KUBELETCONFIG CRD TO EDIT KUBELET PARAMETERS                | 8         |
| 2.3. CONTROL PLANE NODE SIZING  | 10        |
| 2.4. RECOMMENDED ETCD PRACTICES   | 10        |
| 2.5. DEFRAGMENTING ETCD DATA  | 11        |
| 2.6. OPENSIFT CONTAINER PLATFORM INFRASTRUCTURE COMPONENTS                  | 14        |
| 2.7. MOVING THE MONITORING SOLUTION   | 14        |
| 2.8. MOVING THE DEFAULT REGISTRY  | 15        |
| 2.9. MOVING THE ROUTER  | 17        |
| 2.10. INFRASTRUCTURE NODE SIZING  | 18        |
| 2.11. ADDITIONAL RESOURCES  | 19        |
| <b>CHAPTER 3. RECOMMENDED CLUSTER SCALING PRACTICES</b> .....               | <b>20</b> |
| 3.1. RECOMMENDED PRACTICES FOR SCALING THE CLUSTER                          | 20        |
| 3.2. MODIFYING A MACHINE SET  | 20        |
| 3.3. ABOUT MACHINE HEALTH CHECKS  | 21        |
| 3.3.1. Limitations when deploying machine health checks                     | 22        |
| 3.4. SAMPLE MACHINEHEALTHCHECK RESOURCE                                     | 22        |
| 3.4.1. Short-circuiting machine health check remediation                    | 23        |
| 3.4.1.1. Setting maxUnhealthy by using an absolute value                    | 24        |
| 3.4.1.2. Setting maxUnhealthy by using percentages                          | 24        |
| 3.5. CREATING A MACHINEHEALTHCHECK RESOURCE                                 | 24        |
| <b>CHAPTER 4. USING THE NODE TUNING OPERATOR</b> .....                      | <b>25</b> |
| 4.1. ABOUT THE NODE TUNING OPERATOR   | 25        |
| 4.2. ACCESSING AN EXAMPLE NODE TUNING OPERATOR SPECIFICATION                | 25        |
| 4.3. DEFAULT PROFILES SET ON A CLUSTER                                      | 26        |
| 4.4. VERIFYING THAT THE TUNED PROFILES ARE APPLIED                          | 27        |
| 4.5. CUSTOM TUNING SPECIFICATION  | 28        |
| 4.6. CUSTOM TUNING EXAMPLE  | 32        |
| 4.7. SUPPORTED TUNED DAEMON PLUG-INS  | 33        |
| <b>CHAPTER 5. USING CLUSTER LOADER</b> .....                                | <b>35</b> |
| 5.1. INSTALLING CLUSTER LOADER  | 35        |
| 5.2. RUNNING CLUSTER LOADER   | 35        |
| 5.3. CONFIGURING CLUSTER LOADER   | 35        |
| 5.3.1. Example Cluster Loader configuration file                            | 36        |
| 5.3.2. Configuration fields   | 37        |
| 5.4. KNOWN ISSUES   | 40        |
| <b>CHAPTER 6. USING CPU MANAGER</b> .....                                   | <b>41</b> |
| 6.1. SETTING UP CPU MANAGER   | 41        |
| <b>CHAPTER 7. USING TOPOLOGY MANAGER</b> .....                              | <b>46</b> |
| 7.1. TOPOLOGY MANAGER POLICIES  | 46        |
| 7.2. SETTING UP TOPOLOGY MANAGER  | 47        |
| 7.3. POD INTERACTIONS WITH TOPOLOGY MANAGER POLICIES                        | 47        |
| <b>CHAPTER 8. SCALING THE CLUSTER MONITORING OPERATOR</b> .....             | <b>49</b> |

|  |           |
|--|-----------|
| 8.1. PROMETHEUS DATABASE STORAGE REQUIREMENTS  | 49        |
| 8.2. CONFIGURING CLUSTER MONITORING  | 50        |
| <b>CHAPTER 9. PLANNING YOUR ENVIRONMENT ACCORDING TO OBJECT MAXIMUMS</b> .....                             | <b>52</b> |
| 9.1. OPENSIFT CONTAINER PLATFORM TESTED CLUSTER MAXIMUMS FOR MAJOR RELEASES                                | 52        |
| 9.2. OPENSIFT CONTAINER PLATFORM TESTED CLUSTER MAXIMUMS   | 53        |
| 9.3. OPENSIFT CONTAINER PLATFORM ENVIRONMENT AND CONFIGURATION ON WHICH THE CLUSTER<br>MAXIMUMS ARE TESTED | 55        |
| 9.4. HOW TO PLAN YOUR ENVIRONMENT ACCORDING TO TESTED CLUSTER MAXIMUMS                                     | 55        |
| 9.5. HOW TO PLAN YOUR ENVIRONMENT ACCORDING TO APPLICATION REQUIREMENTS                                    | 56        |
| <b>CHAPTER 10. OPTIMIZING STORAGE</b> .....  | <b>60</b> |
| 10.1. AVAILABLE PERSISTENT STORAGE OPTIONS   | 60        |
| 10.2. RECOMMENDED CONFIGURABLE STORAGE TECHNOLOGY  | 61        |
| 10.2.1. Specific application storage recommendations   | 61        |
| 10.2.1.1. Registry   | 62        |
| 10.2.1.2. Scaled registry  | 62        |
| 10.2.1.3. Metrics  | 62        |
| 10.2.1.4. Logging  | 63        |
| 10.2.1.5. Applications   | 63        |
| 10.2.2. Other specific application storage recommendations   | 63        |
| 10.3. DATA STORAGE MANAGEMENT  | 63        |
| <b>CHAPTER 11. OPTIMIZING ROUTING</b> .....  | <b>65</b> |
| 11.1. BASELINE INGRESS CONTROLLER (ROUTER) PERFORMANCE   | 65        |
| 11.2. INGRESS CONTROLLER (ROUTER) PERFORMANCE OPTIMIZATIONS  | 66        |
| <b>CHAPTER 12. OPTIMIZING NETWORKING</b> .....   | <b>67</b> |
| 12.1. OPTIMIZING THE MTU FOR YOUR NETWORK  | 67        |
| 12.2. RECOMMENDED PRACTICES FOR INSTALLING LARGE SCALE CLUSTERS  | 68        |
| 12.3. IMPACT OF IPSEC  | 68        |
| <b>CHAPTER 13. WHAT HUGE PAGES DO AND HOW THEY ARE CONSUMED BY APPLICATIONS</b> .....                      | <b>69</b> |
| 13.1. WHAT HUGE PAGES DO   | 69        |
| 13.2. HOW HUGE PAGES ARE CONSUMED BY APPS  | 69        |
| 13.3. CONFIGURING HUGE PAGES   | 70        |
| 13.3.1. At boot time   | 70        |
| <b>CHAPTER 14. PERFORMANCE ADDON OPERATOR FOR LOW LATENCY NODES</b> .....                                  | <b>73</b> |
| 14.1. UNDERSTANDING LOW LATENCY  | 73        |
| 14.1.1. About hyperthreading for low latency and real-time applications                                    | 73        |
| 14.2. INSTALLING THE PERFORMANCE ADDON OPERATOR  | 74        |
| 14.2.1. Installing the Operator using the CLI  | 74        |
| 14.2.2. Installing the Performance Addon Operator using the web console                                    | 75        |
| 14.3. UPGRADING PERFORMANCE ADDON OPERATOR   | 76        |
| 14.3.1. About upgrading Performance Addon Operator   | 76        |
| 14.3.1.1. How Performance Addon Operator upgrades affect your cluster                                      | 77        |
| 14.3.1.2. Upgrading Performance Addon Operator to the next minor version                                   | 77        |
| 14.3.1.3. Upgrading Performance Addon Operator when previously installed to a specific namespace           | 77        |
| 14.3.2. Monitoring upgrade status  | 78        |
| 14.4. PROVISIONING REAL-TIME AND LOW LATENCY WORKLOADS   | 79        |
| 14.4.1. Known limitations for real-time  | 79        |
| 14.4.2. Provisioning a worker with real-time capabilities  | 79        |
| 14.4.3. Verifying the real-time kernel installation  | 80        |
| 14.4.4. Creating a workload that works in real-time  | 80        |

|   |     |
|---|-----|
| 14.4.5. Creating a pod with a QoS class of Guaranteed                               | 81  |
| 14.4.6. Optional: Disabling CPU load balancing for DPDK                             | 82  |
| 14.4.7. Assigning a proper node selector  | 82  |
| 14.4.8. Scheduling a workload onto a worker with real-time capabilities             | 83  |
| 14.4.9. Managing device interrupt processing for guaranteed pod isolated CPUs       | 83  |
| 14.4.9.1. Disabling global device interrupts handling in Performance Addon Operator | 83  |
| 14.4.9.2. Disabling interrupt processing for individual pods                        | 84  |
| 14.4.10. Upgrading the performance profile to use device interrupt processing       | 84  |
| 14.4.10.1. Supported API Versions   | 84  |
| 14.4.10.1.1. Upgrading Performance Addon Operator API from v1alpha1 to v1           | 84  |
| 14.4.10.1.2. Upgrading Performance Addon Operator API from v1alpha1 or v1 to v2     | 84  |
| 14.4.11. Configuring a node for IRQ dynamic load balancing                          | 84  |
| 14.4.12. Configuring hyperthreading for a cluster                                   | 87  |
| 14.4.12.1. Disabling hyperthreading for low latency applications                    | 89  |
| 14.5. CONFIGURING HUGE PAGES  | 89  |
| 14.6. ALLOCATING MULTIPLE HUGE PAGE SIZES   | 91  |
| 14.7. TUNING NODES FOR LOW LATENCY WITH THE PERFORMANCE PROFILE                     | 91  |
| 14.7.1. Partitioning the CPUs   | 92  |
| 14.8. PERFORMING END-TO-END TESTS FOR PLATFORM VERIFICATION                         | 92  |
| 14.8.1. Prerequisites   | 93  |
| 14.8.2. Running the tests   | 94  |
| 14.8.2.1. Running the latency tests   | 94  |
| 14.8.3. Image parameters  | 96  |
| 14.8.3.1. Ginkgo parameters   | 96  |
| 14.8.3.2. Available features  | 96  |
| 14.8.4. Dry run   | 97  |
| 14.8.5. Disconnected mode   | 97  |
| 14.8.5.1. Mirroring the images to a custom registry accessible from the cluster     | 97  |
| 14.8.5.2. Instruct the tests to consume those images from a custom registry         | 97  |
| 14.8.5.3. Mirroring to the cluster internal registry                                | 97  |
| 14.8.5.4. Mirroring a different set of images                                       | 99  |
| 14.8.6. Discovery mode  | 99  |
| 14.8.6.1. Required environment configuration prerequisites                          | 100 |
| 14.8.6.2. Limiting the nodes used during tests                                      | 101 |
| 14.8.6.3. Using a single performance profile  | 101 |
| 14.8.6.4. Disabling the performance profile cleanup                                 | 101 |
| 14.8.7. Troubleshooting   | 102 |
| 14.8.8. Test reports  | 102 |
| 14.8.8.1. JUnit test output   | 102 |
| 14.8.8.2. Test failure report   | 102 |
| 14.8.8.3. A note on podman  | 102 |
| 14.8.8.4. Running on OpenShift Container Platform 4.4                               | 102 |
| 14.8.8.5. Using a single performance profile  | 103 |
| 14.8.9. Impacts on the cluster  | 103 |
| 14.8.9.1. SCTP  | 103 |
| 14.8.9.2. XT_U32  | 104 |
| 14.8.9.3. SR-IOV  | 104 |
| 14.8.9.4. PTP   | 104 |
| 14.8.9.5. Performance   | 104 |
| 14.8.9.6. DPDK  | 104 |
| 14.8.9.7. Cleaning up   | 104 |
| 14.9. DEBUGGING LOW LATENCY CNF TUNING STATUS                                       | 104 |
| 14.9.1. Machine config pools  | 105 |

|   |     |
|---|-----|
| 14.10. COLLECTING LOW LATENCY TUNING DEBUGGING DATA FOR RED HAT SUPPORT | 107 |
| 14.10.1. About the must-gather tool                                     | 107 |
| 14.10.2. About collecting low latency tuning data                       | 107 |
| 14.10.3. Gathering data about specific features                         | 108 |





# CHAPTER 1. RECOMMENDED PRACTICES FOR INSTALLING LARGE CLUSTERS

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

## 1.1. RECOMMENDED PRACTICES FOR INSTALLING LARGE SCALE CLUSTERS

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

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

The default cluster network **cidr 10.128.0.0/14** cannot be used if the cluster size is more than 500 nodes. It must be set to **10.128.0.0/12** or **10.128.0.0/10** to get to larger node counts beyond 500 nodes.

## CHAPTER 2. RECOMMENDED HOST PRACTICES

This topic provides recommended host practices for OpenShift Container Platform.

### 2.1. RECOMMENDED NODE HOST PRACTICES

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

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

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



#### IMPORTANT

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



#### NOTE

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

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

```
kubeletConfig:
  podsPerCore: 10
```

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

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

```
kubeletConfig:
  maxPods: 250
```

## 2.2. CREATING A KUBELETCONFIG CRD TO EDIT KUBELET PARAMETERS

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

### Procedure

1. Run:

```
$ oc get machineconfig
```

This provides a list of the available machine configuration objects you can select. By default, the two kubelet-related configs are **01-master-kubelet** and **01-worker-kubelet**.

2. To check the current value of max pods per node, run:

```
# oc describe node <node-ip> | grep Allocatable -A6
```

Look for **value: pods: <value>**.

For example:

```
# oc describe node ip-172-31-128-158.us-east-2.compute.internal | grep Allocatable -A6
```

### Example output

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

3. To set the max pods per node on the worker nodes, create a custom resource file that contains the kubelet configuration. For example, **change-maxPods-cr.yaml**:

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

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

```

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

```

a. Run:

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

b. Run:

```
$ oc get kubeletconfig
```

This should return **set-max-pods**.

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

4. Check for **maxPods** changing for the worker nodes:

```
$ oc describe node
```

a. Verify the change by running:

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

This should show a status of **True** and **type:Success**

## Procedure

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

1. Run:

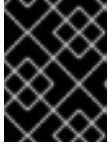
```
$ oc edit machineconfigpool worker
```

2. Set **maxUnavailable** to the desired value.

```

spec:
  maxUnavailable: <node_count>

```

**IMPORTANT**

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

## 2.3. CONTROL PLANE NODE SIZING

The control plane node resource requirements depend on the number of nodes in the cluster. The following control plane node size recommendations are based on the results of control plane density focused testing.

| Number of worker nodes | CPU cores | Memory (GB) |
|------------------------|-----------|-------------|
| 25                     | 4         | 16          |
| 100                    | 8         | 32          |
| 250                    | 16        | 96          |

**IMPORTANT**

Because you cannot modify the control plane node size in a running OpenShift Container Platform 4.7 cluster, you must estimate your total node count and use the suggested control plane node size during installation.

**NOTE**

In OpenShift Container Platform 4.7, half of a CPU core (500 millicore) is now reserved by the system by default compared to OpenShift Container Platform 3.11 and previous versions. The sizes are determined taking that into consideration.

## 2.4. RECOMMENDED ETCD PRACTICES

For large and dense clusters, etcd can suffer from poor performance if the key space grows excessively large and exceeds the space quota. Periodic maintenance of etcd, including defragmentation, must be performed to free up space in the data store. It is highly recommended that you monitor Prometheus for etcd metrics and defragment it when required before etcd raises a cluster-wide alarm that puts the cluster into a maintenance mode, which only accepts key reads and deletes. Some of the key metrics to monitor are **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, and **etcd\_debugging\_mvcc\_db\_total\_size\_in\_bytes** which shows the database size including free space waiting for defragmentation. Instructions on defragging etcd can be found in the **Defragmenting etcd data** section.

Etcd writes data to disk, so its performance strongly depends on disk performance. Etcd persists proposals on disk. Slow disks and disk activity from other processes might cause long fsync latencies, causing etcd to miss heartbeats, inability to commit new proposals to the disk on time, which can cause request timeouts and temporary leader loss. It is highly recommended to run etcd on machines backed by SSD/NVMe disks with low latency and high throughput.

Some of the 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. **etcd\_disk\_wal\_fsync\_duration\_seconds\_bucket** reports the etcd disk fsync duration, **etcd\_server\_leader\_changes\_seen\_total** reports the leader changes. To rule out a slow disk and confirm that the disk is reasonably fast, 99th percentile of the **etcd\_disk\_wal\_fsync\_duration\_seconds\_bucket** should be less than 10ms.

Fio, a I/O benchmarking tool can be used to validate the hardware for etcd before or after creating the OpenShift cluster. Run fio and analyze the results:

Assuming container runtimes like podman or docker are installed on the machine under test and the path etcd writes the data exists - /var/lib/etcd, run:

### Procedure

Run the following if using podman:

```
$ sudo podman run --volume /var/lib/etcd:/var/lib/etcd:Z quay.io/openshift-scale/etcd-perf
```

Alternatively, run the following if using docker:

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

The output reports whether the disk is fast enough to host etcd by comparing the 99th percentile of the fsync metric captured from the run to see if it is less than 10ms.

Etcd replicates the requests among all the members, so its performance strongly depends on network input/output (IO) latency. High network latencies result in etcd heartbeats taking longer than the election timeout, which leads to 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. **histogram\_quantile(0.99, rate(etcd\_network\_peer\_round\_trip\_time\_seconds\_bucket[2m]))** reports the round trip time for etcd to finish replicating the client requests between the members; it should be less than 50 ms.

## 2.5. DEFRAGMENTING ETCD DATA

Manual defragmentation must be performed periodically to reclaim disk space after etcd history compaction and other events cause disk fragmentation.

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.

Because etcd writes data to disk, its performance strongly depends on disk performance. Consider defragmenting etcd every month, twice a month, or as needed for your cluster. You can also monitor the **etcd\_db\_total\_size\_in\_bytes** metric to determine whether defragmentation is necessary.



### WARNING

Defragmenting etcd is a blocking action. The etcd member will not response 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

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

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

### Example output

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

- 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.us-west-1.compute.internal 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**.

- 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
```

## 2.6. OPENSIFT CONTAINER PLATFORM INFRASTRUCTURE COMPONENTS

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

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

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

## 2.7. MOVING THE MONITORING SOLUTION

By default, the Prometheus Cluster Monitoring stack, which contains Prometheus, Grafana, and AlertManager, is deployed to provide cluster monitoring. It is managed by the Cluster Monitoring Operator. To move its components to different machines, you create and apply a custom config map.

### Procedure

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

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |+
    alertmanagerMain:
      nodeSelector:
```

```

    node-role.kubernetes.io/infra: ""
prometheusK8s:
  nodeSelector:
    node-role.kubernetes.io/infra: ""
prometheusOperator:
  nodeSelector:
    node-role.kubernetes.io/infra: ""
grafana:
  nodeSelector:
    node-role.kubernetes.io/infra: ""
k8sPrometheusAdapter:
  nodeSelector:
    node-role.kubernetes.io/infra: ""
kubeStateMetrics:
  nodeSelector:
    node-role.kubernetes.io/infra: ""
telemeterClient:
  nodeSelector:
    node-role.kubernetes.io/infra: ""
openshiftStateMetrics:
  nodeSelector:
    node-role.kubernetes.io/infra: ""
thanosQuerier:
  nodeSelector:
    node-role.kubernetes.io/infra: ""

```

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

2. Apply the new config map:

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

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

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

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

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

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

## 2.8. MOVING THE DEFAULT REGISTRY

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

### Prerequisites

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

### Procedure

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

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

### Example output

```
apiVersion: imageregistry.operator.openshift.io/v1
kind: Config
metadata:
  creationTimestamp: 2019-02-05T13:52:05Z
  finalizers:
  - imageregistry.operator.openshift.io/finalizer
  generation: 1
  name: cluster
  resourceVersion: "56174"
  selfLink: /apis/imageregistry.operator.openshift.io/v1/configs/cluster
  uid: 36fd3724-294d-11e9-a524-12fjee2931b
spec:
  httpSecret: d9a012ccd117b1e6616ceccb2c3bb66a5fed1b5e481623
  logging: 2
  managementState: Managed
  proxy: {}
  replicas: 1
  requests:
    read: {}
    write: {}
  storage:
    s3:
      bucket: image-registry-us-east-1-c92e88cad85b48ec8b312344dff03c82-392c
      region: us-east-1
status:
...
```

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

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

3. Add the following lines of text the **spec** section of the object:

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

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

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

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

- b. Confirm the node has the label you specified:

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

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

## 2.9. MOVING THE ROUTER

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

### Prerequisites

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

### Procedure

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

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

The command output resembles the following text:

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

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

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

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

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

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

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

#### Example output

```

NAME                                READY   STATUS    RESTARTS   AGE   IP           NODE
NOMINATED NODE READINESS GATES
router-default-86798b4b5d-bdlvd 1/1     Running   0          28s   10.130.2.4   ip-10-0-217-226.ec2.internal <none> <none>
router-default-955d875f4-255g8 0/1     Terminating 0        19h   10.129.2.4   ip-10-0-148-172.ec2.internal <none> <none>

```

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

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

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

1 1 Specify the **<node\_name>** that you obtained from the pod list.

#### Example output

```

NAME                                STATUS ROLES   AGE   VERSION
ip-10-0-217-226.ec2.internal Ready  infra,worker 17h   v1.20.0

```

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

## 2.10. INFRASTRUCTURE NODE SIZING

The infrastructure node resource requirements depend on the cluster age, nodes, and objects in the cluster, as these factors can lead to an increase in the number of metrics or time series in Prometheus. The following infrastructure node size recommendations are based on the results of cluster maximums and control plane density focused testing.

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



## IMPORTANT

These sizing recommendations are based on scale tests, which create a large number of objects across the cluster. These tests include reaching some of the cluster maximums. In the case of 250 and 500 node counts on a OpenShift Container Platform 4.7 cluster, these maximums are 10000 namespaces with 61000 pods, 10000 deployments, 181000 secrets, 400 config maps, and so on. Prometheus is a highly memory intensive application; the resource usage depends on various factors including the number of nodes, objects, the Prometheus metrics scraping interval, metrics or time series, and the age of the cluster. The disk size also depends on the retention period. You must take these factors into consideration and size them accordingly.

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



## NOTE

In OpenShift Container Platform 4.7, half of a CPU core (500 millicore) is now reserved by the system by default compared to OpenShift Container Platform 3.11 and previous versions. This influences the stated sizing recommendations.

## 2.11. ADDITIONAL RESOURCES

- [OpenShift Container Platform cluster maximums](#)

## CHAPTER 3. RECOMMENDED CLUSTER SCALING PRACTICES



### IMPORTANT

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.

### 3.1. RECOMMENDED PRACTICES FOR SCALING THE CLUSTER

When scaling up the cluster to higher node counts:

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



### NOTE

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

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

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



### NOTE

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

### 3.2. MODIFYING A MACHINE SET

To make changes to a machine set, edit the **MachineSet** YAML. Then, remove all machines associated with the machine set by deleting each machine or scaling down the machine set to **0** replicas. Then, scale the replicas back to the desired number. Changes you make to a machine set do not affect existing machines.



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



## NOTE

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

## Prerequisites

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

## Procedure

1. Edit the machine set:

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

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

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

Or:

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

Wait for the machines to be removed.

3. Scale up the machine set as needed:

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

Or:

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

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

## 3.3. ABOUT MACHINE HEALTH CHECKS

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

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

**NOTE**

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

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

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

**NOTE**

Consider the timeouts carefully, accounting for workloads and requirements.

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

To stop the check, remove the resource.

### 3.3.1. Limitations when deploying machine health checks

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

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

## 3.4. SAMPLE MACHINEHEALTHCHECK RESOURCE

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

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineHealthCheck
metadata:
  name: example 1
  namespace: openshift-machine-api
spec:
  selector:
```

```

matchLabels:
  machine.openshift.io/cluster-api-machine-role: <role> 2
  machine.openshift.io/cluster-api-machine-type: <role> 3
  machine.openshift.io/cluster-api-machineset: <cluster_name>-<label>-<zone> 4
unhealthyConditions:
- type: "Ready"
  timeout: "300s" 5
  status: "False"
- type: "Ready"
  timeout: "300s" 6
  status: "Unknown"
maxUnhealthy: "40%" 7
nodeStartupTimeout: "10m" 8

```

- 1 Specify the name of the machine health check to deploy.
- 2 3 Specify a label for the machine pool that you want to check.
- 4 Specify the machine set to track in **<cluster\_name>-<label>-<zone>** format. For example, **prod-node-us-east-1a**.
- 5 6 Specify the timeout duration for a node condition. If a condition is met for the duration of the timeout, the machine will be remediated. Long timeouts can result in long periods of downtime for a workload on an unhealthy machine.
- 7 Specify the amount of unhealthy machines allowed in the targeted pool. This can be set as a percentage or an integer.
- 8 Specify the timeout duration that a machine health check must wait for a node to join the cluster before a machine is determined to be unhealthy.



#### NOTE

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

### 3.4.1. Short-circuiting machine health check remediation

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

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



#### IMPORTANT

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

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

#### 3.4.1.1. Setting maxUnhealthy by using an absolute value

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

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

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

#### 3.4.1.2. Setting maxUnhealthy by using percentages

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

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

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

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



#### NOTE

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

## 3.5. CREATING A MACHINEHEALTHCHECK RESOURCE

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

### Prerequisites

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

### Procedure

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

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

## CHAPTER 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. 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 is part of a standard OpenShift Container Platform installation in version 4.1 and later.

### 4.2. ACCESSING AN EXAMPLE NODE TUNING OPERATOR SPECIFICATION

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

#### Procedure

1. Run:

```
$ oc get Tuned/default -o yaml -n openshift-cluster-node-tuning-operator
```

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



#### WARNING

While in certain situations the support for pod labels can be a convenient way of automatically delivering required tuning, this practice is discouraged and strongly advised against, especially in large-scale clusters. The default Tuned CR ships without pod label matching. If a custom profile is created with pod label matching, then the functionality will be enabled at that time. The pod label functionality might 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:
    - name: "openshift"
      data: |
        [main]
        summary=Optimize systems running OpenShift (parent profile)
        include=${f:virt_check:virtual-guest:throughput-performance}

        [selinux]
        avc_cache_threshold=8192

        [net]
        nf_conntrack_hashsize=131072

        [sysctl]
        net.ipv4.ip_forward=1
        kernel.pid_max=>4194304
        net.netfilter.nf_conntrack_max=1048576
        net.ipv4.conf.all.arp_announce=2
        net.ipv4.neigh.default.gc_thresh1=8192
        net.ipv4.neigh.default.gc_thresh2=32768
        net.ipv4.neigh.default.gc_thresh3=65536
        net.ipv6.neigh.default.gc_thresh1=8192
        net.ipv6.neigh.default.gc_thresh2=32768
        net.ipv6.neigh.default.gc_thresh3=65536
        vm.max_map_count=262144

        [sysfs]
        /sys/module/nvme_core/parameters/io_timeout=4294967295
        /sys/module/nvme_core/parameters/max_retries=10

    - name: "openshift-control-plane"
      data: |
        [main]
        summary=Optimize systems running OpenShift control plane
        include=openshift

        [sysctl]
        # ktune sysctl settings, maximizing i/o throughput
        #
        # Minimal preemption granularity for CPU-bound tasks:
        # (default: 1 msec# (1 + ilog(ncpus)), units: nanoseconds)
        kernel.sched_min_granularity_ns=10000000
        # The total time the scheduler will consider a migrated process
        # "cache hot" and thus less likely to be re-migrated
        # (system default is 500000, i.e. 0.5 ms)
        kernel.sched_migration_cost_ns=5000000

```

```

# SCHED_OTHER wake-up granularity.
#
# Preemption granularity when tasks wake up. Lower the value to
# improve wake-up latency and throughput for latency critical tasks.
kernel.sched_wakeup_granularity_ns=4000000

- name: "openshift-node"
data: |
  [main]
  summary=Optimize systems running OpenShift nodes
  include=openshift

  [sysctl]
  net.ipv4.tcp_fastopen=3
  fs.inotify.max_user_watches=65536
  fs.inotify.max_user_instances=8192

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

```

## 4.4. VERIFYING THAT THE TUNED PROFILES ARE APPLIED

Use this procedure to check which Tuned profiles are applied on every node.

### Procedure

1. Check which Tuned pods are running on each node:

```
$ oc get pods -n openshift-cluster-node-tuning-operator -o wide
```

### Example output

```

NAME                                READY STATUS RESTARTS AGE IP          NODE
NOMINATED NODE READINESS GATES
cluster-node-tuning-operator-599489d4f7-k4hw4 1/1 Running 0      6d2h 10.129.0.76
ip-10-0-145-113.eu-west-3.compute.internal <none> <none>
tuned-2jkzp                                1/1 Running 1      6d3h 10.0.145.113 ip-10-0-145-
113.eu-west-3.compute.internal <none> <none>
tuned-g9mxx                                1/1 Running 1      6d3h 10.0.147.108 ip-10-0-
147-108.eu-west-3.compute.internal <none> <none>
tuned-kbxsh                                1/1 Running 1      6d3h 10.0.132.143 ip-10-0-132-
143.eu-west-3.compute.internal <none> <none>
tuned-kn9x6                                1/1 Running 1      6d3h 10.0.163.177 ip-10-0-163-
177.eu-west-3.compute.internal <none> <none>
tuned-vvwxw                                1/1 Running 1      6d3h 10.0.131.87 ip-10-0-131-

```

```
87.eu-west-3.compute.internal <none> <none>
tuned-zqrwq 1/1 Running 1 6d3h 10.0.161.51 ip-10-0-161-
51.eu-west-3.compute.internal <none> <none>
```

2. Extract the profile applied from each pod and match them against the previous list:

```
$ for p in `oc get pods -n openshift-cluster-node-tuning-operator -l openshift-app=tuned -
o=jsonpath='{range .items[*]}{.metadata.name} {end}`; do printf "\n*** $p ***\n" ; oc logs
pod/$p -n openshift-cluster-node-tuning-operator | grep applied; done
```

### Example output

```
*** tuned-2jkzp ***
2020-07-10 13:53:35,368 INFO tuned.daemon.daemon: static tuning from profile
'openshift-control-plane' applied

*** tuned-g9mkx ***
2020-07-10 14:07:17,089 INFO tuned.daemon.daemon: static tuning from profile
'openshift-node' applied
2020-07-10 15:56:29,005 INFO tuned.daemon.daemon: static tuning from profile
'openshift-node-es' applied
2020-07-10 16:00:19,006 INFO tuned.daemon.daemon: static tuning from profile
'openshift-node' applied
2020-07-10 16:00:48,989 INFO tuned.daemon.daemon: static tuning from profile
'openshift-node-es' applied

*** tuned-kbxsh ***
2020-07-10 13:53:30,565 INFO tuned.daemon.daemon: static tuning from profile
'openshift-node' applied
2020-07-10 15:56:30,199 INFO tuned.daemon.daemon: static tuning from profile
'openshift-node-es' applied

*** tuned-kn9x6 ***
2020-07-10 14:10:57,123 INFO tuned.daemon.daemon: static tuning from profile
'openshift-node' applied
2020-07-10 15:56:28,757 INFO tuned.daemon.daemon: static tuning from profile
'openshift-node-es' applied

*** tuned-vvxwx ***
2020-07-10 14:11:44,932 INFO tuned.daemon.daemon: static tuning from profile
'openshift-control-plane' applied

*** tuned-zqrwq ***
2020-07-10 14:07:40,246 INFO tuned.daemon.daemon: static tuning from profile
'openshift-control-plane' applied
```

## 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
```

- 
- 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**.

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

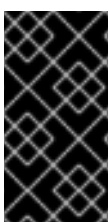
```
- label: <label_name> 1
  value: <label_value> 2
  type: <label_type> 3
  <match> 4
```

- 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.

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



### IMPORTANT

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

### Example: node or pod label based matching

```

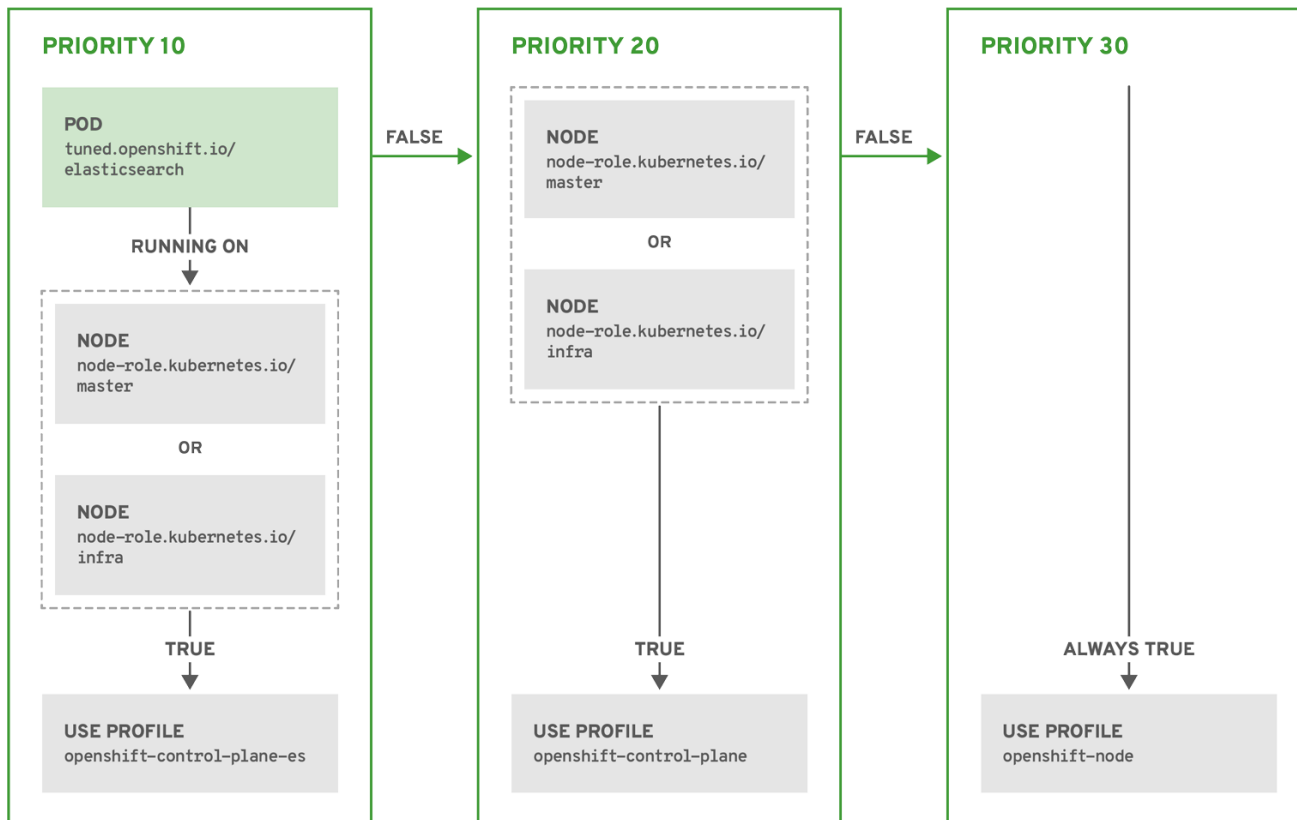
- match:
  - label: tuned.openshift.io/elasticsearch
    match:
      - label: node-role.kubernetes.io/master
      - label: node-role.kubernetes.io/infra
    type: pod
  priority: 10
  profile: openshift-control-plane-es
- match:
  - label: node-role.kubernetes.io/master
  - label: node-role.kubernetes.io/infra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

```

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

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

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



OPENSIFT\_10\_0319

### Example: machine config pool based matching

```

apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
  name: openshift-node-custom
  namespace: openshift-cluster-node-tuning-operator
spec:
  profile:
  - data: |
    [main]
    summary=Custom OpenShift node profile with an additional kernel parameter
    include=openshift-node
    [bootloader]
    cmdline_openshift_node_custom=+skew_tick=1
    name: openshift-node-custom

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

```

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

## 4.6. CUSTOM TUNING EXAMPLE

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. As an administrator, use the following command to create a custom Tuned CR.

### Custom tuning example

```
$ oc create -f <<_EOF_
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
_EOF_
```



#### 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.

## 4.7. SUPPORTED TUNED DAEMON PLUG-INS

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

- audio
- cpu
- disk
- eeepc\_she
- modules
- mounts
- net
- scheduler

- scsi\_host
- selinux
- sysctl
- sysfs
- usb
- video
- vm

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

- bootloader
- script
- systemd

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

## CHAPTER 5. USING CLUSTER LOADER

Cluster Loader is a tool that deploys large numbers of various objects to a cluster, which creates user-defined cluster objects. Build, configure, and run Cluster Loader to measure performance metrics of your OpenShift Container Platform deployment at various cluster states.

### 5.1. INSTALLING CLUSTER LOADER

#### Procedure

1. To pull the container image, run:

```
$ podman pull quay.io/openshift/origin-tests:4.7
```

### 5.2. RUNNING CLUSTER LOADER

#### Prerequisites

- The repository will prompt you to authenticate. The registry credentials allow you to access the image, which is not publicly available. Use your existing authentication credentials from installation.

#### Procedure

1. Execute Cluster Loader using the built-in test configuration, which deploys five template builds and waits for them to complete:

```
$ podman run -v ${LOCAL_KUBECONFIG}:/root/.kube/config:z -i \
quay.io/openshift/origin-tests:4.7 /bin/bash -c 'export KUBECONFIG=/root/.kube/config && \
openshift-tests run-test "[sig-scalability][Feature:Performance] Load cluster \
should populate the cluster [Slow][Serial] [Suite:openshift]'"
```

Alternatively, execute Cluster Loader with a user-defined configuration by setting the environment variable for **VIPERCONFIG**:

```
$ podman run -v ${LOCAL_KUBECONFIG}:/root/.kube/config:z \
-v ${LOCAL_CONFIG_FILE_PATH}:/root/configs:z \
-i quay.io/openshift/origin-tests:4.7 \
/bin/bash -c 'KUBECONFIG=/root/.kube/config VIPERCONFIG=/root/configs/test.yaml \
openshift-tests run-test "[sig-scalability][Feature:Performance] Load cluster \
should populate the cluster [Slow][Serial] [Suite:openshift]'"
```

In this example, **\${LOCAL\_KUBECONFIG}** refers to the path to the **kubeconfig** on your local file system. Also, there is a directory called **\${LOCAL\_CONFIG\_FILE\_PATH}**, which is mounted into the container that contains a configuration file called **test.yaml**. Additionally, if the **test.yaml** references any external template files or podspec files, they should also be mounted into the container.

### 5.3. CONFIGURING CLUSTER LOADER

The tool creates multiple namespaces (projects), which contain multiple templates or pods.

### 5.3.1. Example Cluster Loader configuration file

Cluster Loader's configuration file is a basic YAML file:

```
provider: local ❶
ClusterLoader:
  cleanup: true
  projects:
    - num: 1
      basename: clusterloader-cakephp-mysql
      tuning: default
      ifexists: reuse
      templates:
        - num: 1
          file: cakephp-mysql.json

    - num: 1
      basename: clusterloader-dancer-mysql
      tuning: default
      ifexists: reuse
      templates:
        - num: 1
          file: dancer-mysql.json

    - num: 1
      basename: clusterloader-django-postgresql
      tuning: default
      ifexists: reuse
      templates:
        - num: 1
          file: django-postgresql.json

    - num: 1
      basename: clusterloader-nodejs-mongodb
      tuning: default
      ifexists: reuse
      templates:
        - num: 1
          file: quickstarts/nodejs-mongodb.json

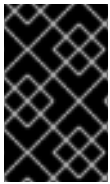
    - num: 1
      basename: clusterloader-rails-postgresql
      tuning: default
      templates:
        - num: 1
          file: rails-postgresql.json

  tuningsets: ❷
    - name: default
      pods:
        stepping: ❸
          stepsize: 5
          pause: 0 s
        rate_limit: ❹
          delay: 0 ms
```



- 1 Optional setting for end-to-end tests. Set to **local** to avoid extra log messages.
- 2 The tuning sets allow rate limiting and stepping, the ability to create several batches of pods while pausing in between sets. Cluster Loader monitors completion of the previous step before continuing.
- 3 Stepping will pause for **M** seconds after each **N** objects are created.
- 4 Rate limiting will wait **M** milliseconds between the creation of objects.

This example assumes that references to any external template files or pod spec files are also mounted into the container.



### IMPORTANT

If you are running Cluster Loader on Microsoft Azure, then you must set the **AZURE\_AUTH\_LOCATION** variable to a file that contains the output of **terraform.azure.auto.tfvars.json**, which is present in the installer directory.

## 5.3.2. Configuration fields

Table 5.1. Top-level Cluster Loader Fields

| Field             | Description   |
|-------------------|---|
| <b>cleanup</b>    | Set to <b>true</b> or <b>false</b> . One definition per configuration. If set to <b>true</b> , <b>cleanup</b> deletes all namespaces (projects) created by Cluster Loader at the end of the test.   |
| <b>projects</b>   | A sub-object with one or many definition(s). Under <b>projects</b> , each namespace to create is defined and <b>projects</b> has several mandatory subheadings.                                     |
| <b>tuningsets</b> | A sub-object with one definition per configuration. <b>tuningsets</b> allows the user to define a tuning set to add configurable timing to project or object creation (pods, templates, and so on). |
| <b>sync</b>       | An optional sub-object with one definition per configuration. Adds synchronization possibilities during object creation.  |

Table 5.2. Fields under **projects**

| Field      | Description   |
|------------|---|
| <b>num</b> | An integer. One definition of the count of how many projects to create. |

| Field             | Description  |
|-------------------|--|
| <b>basename</b>   | A string. One definition of the base name for the project. The count of identical namespaces will be appended to <b>basename</b> to prevent collisions.  |
| <b>tuning</b>     | A string. One definition of what tuning set you want to apply to the objects, which you deploy inside this namespace.  |
| <b>ifexists</b>   | A string containing either <b>reuse</b> or <b>delete</b> . Defines what the tool does if it finds a project or namespace that has the same name of the project or namespace it creates during execution. |
| <b>configmaps</b> | A list of key-value pairs. The key is the config map name and the value is a path to a file from which you create the config map.  |
| <b>secrets</b>    | A list of key-value pairs. The key is the secret name and the value is a path to a file from which you create the secret.  |
| <b>Pods</b>       | A sub-object with one or many definition(s) of pods to deploy.   |
| <b>templates</b>  | A sub-object with one or many definition(s) of templates to deploy.  |

Table 5.3. Fields under **Pods** and **templates**

| Field             | Description   |
|-------------------|---|
| <b>num</b>        | An integer. The number of pods or templates to deploy.  |
| <b>image</b>      | A string. The docker image URL to a repository where it can be pulled.  |
| <b>basename</b>   | A string. One definition of the base name for the template (or pod) that you want to create.                    |
| <b>file</b>       | A string. The path to a local file, which is either a pod spec or template to be created.                       |
| <b>parameters</b> | Key-value pairs. Under <b>parameters</b> , you can specify a list of values to override in the pod or template. |

Table 5.4. Fields under **tuningsets**

| Field            | Description   |
|------------------|---|
| <b>name</b>      | A string. The name of the tuning set which will match the name specified when defining a tuning in a project. |
| <b>pods</b>      | A sub-object identifying the <b>tuningsets</b> that will apply to pods.                                       |
| <b>templates</b> | A sub-object identifying the <b>tuningsets</b> that will apply to templates.                                  |

Table 5.5. Fields under **tuningsets pods** or **tuningsets templates**

| Field             | Description   |
|-------------------|---|
| <b>stepping</b>   | A sub-object. A stepping configuration used if you want to create an object in a step creation pattern. |
| <b>rate_limit</b> | A sub-object. A rate-limiting tuning set configuration to limit the object creation rate.               |

Table 5.6. Fields under **tuningsets pods** or **tuningsets templates, stepping**

| Field           | Description   |
|-----------------|---|
| <b>stepsize</b> | An integer. How many objects to create before pausing object creation.                                  |
| <b>pause</b>    | An integer. How many seconds to pause after creating the number of objects defined in <b>stepsize</b> . |
| <b>timeout</b>  | An integer. How many seconds to wait before failure if the object creation is not successful.           |
| <b>delay</b>    | An integer. How many milliseconds (ms) to wait between creation requests.                               |

Table 5.7. Fields under **sync**

| Field | Description |
|-------|-------------|
|-------|-------------|

| Field            | Description   |
|------------------|---|
| <b>server</b>    | A sub-object with <b>enabled</b> and <b>port</b> fields. The boolean <b>enabled</b> defines whether to start an HTTP server for pod synchronization. The integer <b>port</b> defines the HTTP server port to listen on ( <b>9090</b> by default). |
| <b>running</b>   | A boolean. Wait for pods with labels matching <b>selectors</b> to go into <b>Running</b> state.   |
| <b>succeeded</b> | A boolean. Wait for pods with labels matching <b>selectors</b> to go into <b>Completed</b> state.   |
| <b>selectors</b> | A list of selectors to match pods in <b>Running</b> or <b>Completed</b> states.   |
| <b>timeout</b>   | A string. The synchronization timeout period to wait for pods in <b>Running</b> or <b>Completed</b> states. For values that are not <b>0</b> , use units: [ns us ms s m h].   |

## 5.4. KNOWN ISSUES

- Cluster Loader fails when called without configuration. ([BZ#1761925](#))
- If the **IDENTIFIER** parameter is not defined in user templates, template creation fails with **error: unknown parameter name "IDENTIFIER"**. If you deploy templates, add this parameter to your template to avoid this error:

```
{
  "name": "IDENTIFIER",
  "description": "Number to append to the name of resources",
  "value": "1"
}
```

If you deploy pods, adding the parameter is unnecessary.

## CHAPTER 6. USING CPU MANAGER

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

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

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

### 6.1. SETTING UP CPU MANAGER

#### Procedure

1. Optional: Label a node:

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

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

```
# oc edit machineconfigpool worker
```

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

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

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

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

- 1** Specify a policy:

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

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

5. Create the dynamic kubelet config:

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

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

6. Check for the merged kubelet config:

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

### Example output

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

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

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

### Example output

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

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

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

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

### Example output

```
apiVersion: v1
```

```

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

```

9. Create the pod:

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

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

```
# oc describe pod cpumanager
```

### Example output

```

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

```

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

```

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

```

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

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

### Example output

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

- Check the allowed CPU list for the task:

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

### Example output

```
Cpus_allowed_list: 1
```

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

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

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

### Example output

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

Allocated resources:
```



(Total limits may be over 100 percent, i.e., overcommitted.)

| Resource | Requests    | Limits  |
|----------|-------------|---------|
| -----    | -----       | -----   |
| cpu      | 1440m (96%) | 1 (66%) |

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

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

## CHAPTER 7. USING TOPOLOGY MANAGER

Topology Manager collects hints from the CPU Manager, Device Manager, and other Hint Providers to align pod resources, such as CPU, SR-IOV VFs, and other device resources, for all Quality of Service (QoS) classes on the same non-uniform memory access (NUMA) node.

Topology Manager uses topology information from collected hints to decide if a pod can be accepted or rejected on a node, based on the configured Topology Manager policy and Pod resources requested.

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



### NOTE

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

### 7.1. TOPOLOGY MANAGER POLICIES

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



### NOTE

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

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

#### **none policy**

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

#### **best-effort policy**

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

#### **restricted policy**

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

#### **single-numa-node policy**

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

## 7.2. SETTING UP TOPOLOGY MANAGER

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

### Prerequisites

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

### Procedure

To activate Topology Manager:

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

```
$ oc edit KubeletConfig cpumanager-enabled

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

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

### Additional resources

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

## 7.3. POD INTERACTIONS WITH TOPOLOGY MANAGER POLICIES

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

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

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

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

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

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

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

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

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

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

## CHAPTER 8. 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 system resources, containers and components metrics in one dashboard interface, Grafana.



### IMPORTANT

If you are running cluster monitoring with an attached PVC for Prometheus, you might experience OOM kills during cluster upgrade. When persistent storage is in use for Prometheus, Prometheus memory usage doubles during cluster upgrade and for several hours after upgrade is complete. To avoid the OOM kill issue, allow worker nodes with double the size of memory that was available prior to the upgrade. For example, if you are running monitoring on the minimum recommended nodes, which is 2 cores with 8 GB of RAM, increase memory to 16 GB. For more information, see [BZ#1925061](#).

### 8.1. PROMETHEUS DATABASE STORAGE REQUIREMENTS

Red Hat performed various tests for different scale sizes.



### NOTE

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

**Table 8.1. Prometheus Database storage requirements based on number of nodes/pods in the cluster**

| Number of Nodes | Number of pods | Prometheus storage growth per day | Prometheus storage growth per 15 days | RAM Space (per scale size) | Network (per tsdb chunk) |
|-----------------|----------------|-----------------------------------|---------------------------------------|----------------------------|--------------------------|
| 50              | 1800           | 6.3 GB                            | 94 GB                                 | 6 GB                       | 16 MB                    |
| 100             | 3600           | 13 GB                             | 195 GB                                | 10 GB                      | 26 MB                    |
| 150             | 5400           | 19 GB                             | 283 GB                                | 12 GB                      | 36 MB                    |
| 200             | 7200           | 25 GB                             | 375 GB                                | 14 GB                      | 46 MB                    |

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

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

**NOTE**

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

**Recommendations for OpenShift Container Platform**

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

**8.2. CONFIGURING CLUSTER MONITORING****Procedure**

To increase the storage capacity for Prometheus:

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

```

apiVersion: v1
kind: ConfigMap
data:
  config.yaml: |
    prometheusOperator:
      baseImage: quay.io/coreos/prometheus-operator
      prometheusConfigReloaderBaseImage: quay.io/coreos/prometheus-config-reloader
      configReloaderBaseImage: quay.io/coreos/configmap-reload
      nodeSelector:
        node-role.kubernetes.io/infra: ""
    prometheusK8s:
      retention: {{PROMETHEUS_RETENTION_PERIOD}} 1
      baseImage: openshift/prometheus
      nodeSelector:
        node-role.kubernetes.io/infra: ""
      volumeClaimTemplate:
        spec:
          storageClassName: gp2
          resources:
            requests:
              storage: {{PROMETHEUS_STORAGE_SIZE}} 2
    alertmanagerMain:
      baseImage: openshift/prometheus-alertmanager
      nodeSelector:
        node-role.kubernetes.io/infra: ""
      volumeClaimTemplate:
        spec:
          storageClassName: gp2
          resources:
            requests:
              storage: {{ALERTMANAGER_STORAGE_SIZE}} 3
    nodeExporter:
      baseImage: openshift/prometheus-node-exporter
    kubeRbacProxy:
      baseImage: quay.io/coreos/kube-rbac-proxy

```

```

kubeStateMetrics:
  baseImage: quay.io/coreos/kube-state-metrics
  nodeSelector:
    node-role.kubernetes.io/infra: ""
grafana:
  baseImage: grafana/grafana
  nodeSelector:
    node-role.kubernetes.io/infra: ""
auth:
  baseImage: openshift/oauth-proxy
k8sPrometheusAdapter:
  nodeSelector:
    node-role.kubernetes.io/infra: ""
metadata:
  name: cluster-monitoring-config
namespace: openshift-monitoring

```

- 1 A typical value is **PROMETHEUS\_RETENTION\_PERIOD=15d**. Units are measured in time using one of these suffixes: s, m, h, d.
- 2 A typical value is **PROMETHEUS\_STORAGE\_SIZE=2000Gi**. Storage values can be a plain integer or as a fixed-point integer using one of these suffixes: E, P, T, G, M, K. You can also use the power-of-two equivalents: Ei, Pi, Ti, Gi, Mi, Ki.
- 3 A typical value is **ALERTMANAGER\_STORAGE\_SIZE=20Gi**. Storage values can be a plain integer or as a fixed-point integer using one of these suffixes: E, P, T, G, M, K. You can also use the power-of-two equivalents: Ei, Pi, Ti, Gi, Mi, Ki.

2. Set the values like the retention period and storage sizes.
3. Apply the changes by running:

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

## CHAPTER 9. 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.

### 9.1. OPENSIFT CONTAINER PLATFORM TESTED CLUSTER MAXIMUMS FOR MAJOR RELEASES

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

| Maximum type                                       | 3.x tested maximum                                     | 4.x tested maximum  |
|--|--|---|
| Number of nodes                                    | 2,000  | 2,000   |
| Number of pods <sup>[1]</sup>                      | 150,000  | 150,000   |
| Number of pods per node                            | 250  | 500 <sup>[2]</sup>  |
| Number of pods per core                            | There is no default value.                             | There is no default value.  |
| Number of namespaces <sup>[3]</sup>                | 10,000   | 10,000  |
| Number of builds                                   | 10,000 (Default pod RAM 512 Mi)<br>- Pipeline Strategy | 10,000 (Default pod RAM 512 Mi)<br>- Source-to-Image (S2I) build strategy |
| Number of pods per namespace <sup>[4]</sup>        | 25,000   | 25,000  |
| Number of services <sup>[5]</sup>                  | 10,000   | 10,000  |
| Number of services per namespace                   | 5,000  | 5,000   |
| Number of back-ends per service                    | 5,000  | 5,000   |
| Number of deployments per namespace <sup>[4]</sup> | 2,000  | 2,000   |



1. 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.
2. 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 Container Storage v4 (OCS v4) was able to satisfy the number of pods per node discussed in this document.
3. When there are a large number of active projects, etcd might suffer from poor performance if the key space grows excessively large and exceeds the space quota. Periodic maintenance of etcd, including defragmentation, is highly recommended to free etcd storage.
4. 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.
5. 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.

## 9.2. OPENSIFT CONTAINER PLATFORM TESTED CLUSTER MAXIMUMS

| Maximum type                        | 4.1 and 4.2 tested maximum | 4.3 tested maximum         | 4.4 tested maximum         | 4.5 tested maximum         | 4.6 tested maximum         | 4.7 tested maximum         |
|-------------------------------------|----------------------------|----------------------------|----------------------------|----------------------------|----------------------------|----------------------------|
| Number of nodes                     | 2,000                      | 2,000                      | 250                        | 500                        | 500                        | 500                        |
| Number of pods <sup>[1]</sup>       | 150,000                    | 150,000                    | 62,500                     | 62,500                     | 62,500                     | 62,500                     |
| Number of pods per node             | 250                        | 500                        | 500                        | 500                        | 500                        | 500                        |
| Number of pods per core             | There is no default value. | There is no default value. | There is no default value. | There is no default value. | There is no default value. | There is no default value. |
| Number of namespaces <sup>[2]</sup> | 10,000                     | 10,000                     | 10,000                     | 10,000                     | 10,000                     | 10,000                     |

| Maximum type                            | 4.1 and 4.2 tested maximum                          | 4.3 tested maximum   | 4.4 tested maximum   | 4.5 tested maximum   | 4.6 tested maximum   | 4.7 tested maximum   |
|---|---|--|--|--|--|--|
| Number of builds                        | 10,000 (Default pod RAM 512 Mi) - Pipeline Strategy | 10,000 (Default pod RAM 512 Mi) - Source-to-Image (S2I) build strategy | 10,000 (Default pod RAM 512 Mi) - Source-to-Image (S2I) build strategy | 10,000 (Default pod RAM 512 Mi) - Source-to-Image (S2I) build strategy | 10,000 (Default pod RAM 512 Mi) - Source-to-Image (S2I) build strategy | 10,000 (Default pod RAM 512 Mi) - Source-to-Image (S2I) build strategy |
| Number of pods per namespace [3]        | 25,000  | 25,000   | 25,000   | 25,000   | 25,000   | 25,000   |
| Number of services [4]                  | 10,000  | 10,000   | 10,000   | 10,000   | 10,000   | 10,000   |
| Number of services per namespace        | 5,000   | 5,000  | 5,000  | 5,000  | 5,000  | 5,000  |
| Number of back ends per service         | 5,000   | 5,000  | 5,000  | 5,000  | 5,000  | 5,000  |
| Number of deployments per namespace [3] | 2,000   | 2,000  | 2,000  | 2,000  | 2,000  | 2,000  |

1. 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.
2. 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.
3. 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.
4. 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.

In OpenShift Container Platform 4.7, half of a CPU core (500 millicore) is reserved by the system compared to OpenShift Container Platform 3.11 and previous versions.

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

AWS cloud platform:

| Node                       | Flavor      | vCPU | RAM(GiB) | Disk type | Disk size(GiB) /IOS | Count                        | Region    |
|----------------------------|-------------|------|----------|-----------|---------------------|------------------------------|-----------|
| Master/etcd <sup>[1]</sup> | r5.4xlarge  | 16   | 128      | io1       | 220 / 3000          | 3                            | us-west-2 |
| Infra <sup>[2]</sup>       | m5.12xlarge | 48   | 192      | gp2       | 100                 | 3                            | us-west-2 |
| Workload <sup>[3]</sup>    | m5.4xlarge  | 16   | 64       | gp2       | 500 <sup>[4]</sup>  | 1                            | us-west-2 |
| Worker                     | m5.2xlarge  | 8    | 32       | gp2       | 100                 | 3/25/250 /500 <sup>[5]</sup> | us-west-2 |

1. io1 disks with 3000 IOPS are used for master/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 and performance and scalability tests are executed at the specified node counts.

### 9.4. HOW TO PLAN YOUR ENVIRONMENT ACCORDING TO TESTED CLUSTER MAXIMUMS



## IMPORTANT

Oversubscribing the physical resources on a node affects resource guarantees the Kubernetes scheduler makes during pod placement. Learn what measures you can take to avoid memory swapping.

Some of the tested maximums are stretched only in a single dimension. They will vary when many objects are running on the cluster.

The numbers noted in this documentation are based on Red Hat's test methodology, setup, configuration, and tunings. These numbers can vary based on your own individual setup and environments.

While planning your environment, determine how many pods are expected to fit per node:

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

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

### Example scenario

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

$$2200 / 500 = 4.4$$

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

$$2200 / 20 = 110$$

Where:

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

## 9.5. 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:

```
---
Kind: Template
apiVersion: v1
metadata:
  name: deploymentConfigTemplate
  creationTimestamp:
  annotations:
    description: This template will create a deploymentConfig with 1 replica, 4 env vars and a
service.
  tags: "
objects:
- kind: DeploymentConfig
  apiVersion: v1
  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${IDENTIFIER}
  triggers:
  - type: ConfigChange
  strategy:
    type: Rolling
- kind: Service
  apiVersion: v1
  metadata:
    name: service${IDENTIFIER}
  spec:
    selector:
      name: replicationcontroller${IDENTIFIER}
    ports:
    - name: serviceport${IDENTIFIER}
      protocol: TCP
      port: 80
      targetPort: 8080
    portName: ""
    type: ClusterIP
    sessionAffinity: None
  status:
    loadBalancer: {}
  parameters:
  - name: IDENTIFIER
    description: Number to append to the name of resources
    value: '1'
    required: true
  - name: IMAGE
    description: Image to use for deploymentConfig
    value: gcr.io/google-containers/pause-amd64:3.0
    required: false
  - name: ENV_VALUE
```

```

description: Value to use for environment variables
generate: expression
from: "[A-Za-z0-9]{255}"
required: false
labels:
template: deploymentConfigTemplate

```

The number of application pods that can run in a namespace is dependent on the number of services and the length of the service name when the environment variables are used for service discovery.

**ARG\_MAX** on the system defines the maximum argument length for a new process and it is set to **2097152 KiB** by default. The Kubelet injects environment variables in to each pod scheduled to run in the namespace including:

- **<SERVICE\_NAME>\_SERVICE\_HOST=<IP>**
- **<SERVICE\_NAME>\_SERVICE\_PORT=<PORT>**
- **<SERVICE\_NAME>\_PORT=tcp://<IP>:<PORT>**
- **<SERVICE\_NAME>\_PORT\_<PORT>\_TCP=tcp://<IP>:<PORT>**
- **<SERVICE\_NAME>\_PORT\_<PORT>\_TCP\_PROTO=tcp**
- **<SERVICE\_NAME>\_PORT\_<PORT>\_TCP\_PORT=<PORT>**
- **<SERVICE\_NAME>\_PORT\_<PORT>\_TCP\_ADDR=<ADDR>**

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

## CHAPTER 10. 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.

### 10.1. AVAILABLE PERSISTENT STORAGE OPTIONS

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

**Table 10.1. Available storage options**

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

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



**IMPORTANT**

Currently, CNS is not supported in OpenShift Container Platform 4.7.

## 10.2. RECOMMENDED CONFIGURABLE STORAGE TECHNOLOGY

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

**Table 10.2. Recommended and configurable storage technology**

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

<sup>1</sup> **ReadOnlyMany**

<sup>2</sup> **ReadWriteMany**

<sup>3</sup> Prometheus is the underlying technology used for metrics.

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

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

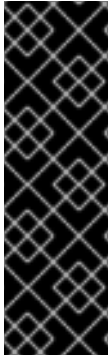
<sup>6</sup> For logging, using any shared storage would be an anti-pattern. One volume per elasticsearch is required.

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

**NOTE**

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

### 10.2.1. 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.

**10.2.1.1. Registry**

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

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

**10.2.1.2. Scaled registry**

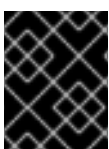
In a scaled/HA OpenShift Container Platform registry cluster deployment:

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

**10.2.1.3. 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.

#### 10.2.1.4. Logging

In an OpenShift Container Platform hosted logging cluster deployment:

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



#### IMPORTANT

Testing shows issues with using the NFS server on RHEL as storage backend for core services. This includes 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.

#### 10.2.1.5. 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.

#### 10.2.2. Other specific application storage recommendations

- OpenShift Container Platform Internal **etcd**: For the best **etcd** reliability, the lowest consistent latency storage technology is preferable.
- It is highly recommended that you use **etcd** with storage that handles serial writes (fsync) quickly, such as NVMe or SSD. Ceph, NFS, and spinning disks are not recommended.
- 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.

### 10.3. DATA STORAGE MANAGEMENT

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

**Table 10.3. Main directories for storing OpenShift Container Platform data**

| Directory                  | Notes  | Sizing  | Expected growth  |
|----------------------------|--|---|--|
| <i>/var/lib/etcd</i>       | Used for etcd storage when storing the database.   | Less than 20 GB.<br><br>Database can grow up to 8 GB.   | Will grow slowly with the environment. Only storing metadata.  |
| Directory                  | Notes  | Sizing  | Expected growth  |
|                            |  |   | Additional 20-25 GB for every additional 8 GB of memory.   |
| <i>/var/lib/containers</i> | This is the mount point for the CRI-O runtime. Storage used for active container runtimes, including pods, and storage of local images. Not used for registry storage.   | 50 GB for a node with 16 GB memory. Note that this sizing should not be used to determine minimum cluster requirements.<br><br>Additional 20-25 GB for every additional 8 GB of memory. | Growth is limited by capacity for running containers.  |
| <i>/var/lib/kubelet</i>    | Ephemeral volume storage for pods. This includes anything external that is mounted into a container at runtime. Includes environment variables, kube secrets, and data volumes not backed by persistent volumes. | Varies  | Minimal if pods requiring storage are using persistent volumes. If using ephemeral storage, this can grow quickly. |
| <i>/var/log</i>            | Log files for all components.  | 10 to 30 GB.  | Log files can grow quickly; size can be managed by growing disks or by using log rotate.                           |

## CHAPTER 11. OPTIMIZING ROUTING

The OpenShift Container Platform HAProxy router scales to optimize performance.

### 11.1. BASELINE INGRESS CONTROLLER (ROUTER) PERFORMANCE

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

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

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

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

In HTTP keep-alive mode scenarios:

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

In HTTP close (no keep-alive) scenarios:

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

| Encryption | LoadBalancerService | HostNetwork |
|------------|---------------------|-------------|
| re-encrypt | 2320                | 2941        |

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

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

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

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

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

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

## 11.2. INGRESS CONTROLLER (ROUTER) PERFORMANCE OPTIMIZATIONS

OpenShift Container Platform no longer supports modifying Ingress Controller deployments by setting environment variables such as **ROUTER\_THREADS**, **ROUTER\_DEFAULT\_TUNNEL\_TIMEOUT**, **ROUTER\_DEFAULT\_CLIENT\_TIMEOUT**, **ROUTER\_DEFAULT\_SERVER\_TIMEOUT**, and **RELOAD\_INTERVAL**.

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

## CHAPTER 12. OPTIMIZING NETWORKING

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

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

VXLAN provides benefits over VLANs, such as an increase in networks from 4096 to over 16 million, and layer 2 connectivity across physical networks. This allows for all pods behind a service to communicate with each other, even if they are running on different systems.

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

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

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

- Evaluate network plug-ins that implement different routing techniques, such as border gateway protocol (BGP).
- Use VXLAN-offload capable network adapters. VXLAN-offload moves the packet checksum calculation and associated CPU overhead off of the system CPU and onto dedicated hardware on the network adapter. This frees up CPU cycles for use by pods and applications, and allows users to utilize the full bandwidth of their network infrastructure.

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

### 12.1. OPTIMIZING THE MTU FOR YOUR NETWORK

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

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

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

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

**NOTE**

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

## 12.2. 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
  machineCIDR: 10.0.0.0/16
  networkType: OpenShiftSDN
  serviceNetwork:
    - 172.30.0.0/16
```

The default cluster network **cidr 10.128.0.0/14** cannot be used if the cluster size is more than 500 nodes. It must be set to **10.128.0.0/12** or **10.128.0.0/10** to get to larger node counts beyond 500 nodes.

## 12.3. 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.

### Additional resources

- [Modifying advanced network configuration parameters](#)
- [Configuration parameters for the OVN-Kubernetes default CNI network provider](#)
- [Configuration parameters for the OpenShift SDN default CNI network provider](#)



## CHAPTER 13. WHAT HUGE PAGES DO AND HOW THEY ARE CONSUMED BY APPLICATIONS

### 13.1. 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.

### 13.2. 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*.

### Additional resources

- [Configuring Transparent Huge Pages](#)

## 13.3. CONFIGURING HUGE PAGES

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

### 13.3.1. At boot time

#### Procedure

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

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

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

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

```

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

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

```

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

3. Create the Tuned **hugepages** profile

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

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

```

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

```

5. Create the machine config pool:

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

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

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



### WARNING

This functionality is currently only supported on Red Hat Enterprise Linux CoreOS (RHCOS) 8.x worker nodes. On Red Hat Enterprise Linux (RHEL) 7.x worker nodes the Tuned **[bootloader]** plug-in is currently not supported.

# CHAPTER 14. PERFORMANCE ADDON OPERATOR FOR LOW LATENCY NODES

## 14.1. 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. Obviously, 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 50ms, 5G is targeted to reach latency numbers of 1ms 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 literally 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 provides a Performance Addon Operator to implement automatic tuning to achieve low latency performance for OpenShift 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, the CPUs that will be reserved for housekeeping, and the CPUs that will be used for running the workloads.

### 14.1.1. 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](#)

**14.2. INSTALLING THE PERFORMANCE ADDON OPERATOR**

Performance Addon Operator provides the ability to enable advanced node performance tunings on a set of nodes. As a cluster administrator, you can install Performance Addon Operator using the OpenShift Container Platform CLI or the web console.

**14.2.1. Installing the Operator using the CLI**

As a cluster administrator, you can install the Operator using the CLI.

**Prerequisites**

- A cluster installed on bare-metal hardware.
- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

**Procedure**

1. Create a namespace for the Performance Addon Operator by completing the following actions:
  - a. Create the following Namespace Custom Resource (CR) that defines the **openshift-performance-addon-operator** namespace, and then save the YAML in the **pao-namespace.yaml** file:

```
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-performance-addon-operator
labels:
  openshift.io/run-level: "1"
```

- b. Create the namespace by running the following command:

```
$ oc create -f pao-namespace.yaml
```

2. Install the Performance Addon Operator in the namespace you created in the previous step by creating the following objects:
  - a. Create the following **OperatorGroup** CR and save the YAML in the **pao-operatorgroup.yaml** file:

```

apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: openshift-performance-addon-operator
  namespace: openshift-performance-addon-operator

```

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

```
$ oc create -f pao-operatorgroup.yaml
```

- c. Run the following command to get the **channel** value required for the next step.

```
$ oc get packagemanifest performance-addon-operator -n openshift-marketplace -o
jsonpath='{.status.defaultChannel}'
```

### Example output

```
4.7
```

- d. Create the following Subscription CR and save the YAML in the **pao-sub.yaml** file:

### Example Subscription

```

apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-performance-addon-operator-subscription
  namespace: openshift-performance-addon-operator
spec:
  channel: "<channel>" 1
  name: performance-addon-operator
  source: redhat-operators 2
  sourceNamespace: openshift-marketplace

```

- 1** Specify the value from you obtained in the previous step for the **.status.defaultChannel** parameter.

- 2** You must specify the **redhat-operators** value.

- e. Create the Subscription object by running the following command:

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

- f. Change to the **openshift-performance-addon-operator** project:

```
$ oc project openshift-performance-addon-operator
```

## 14.2.2. Installing the Performance Addon Operator using the web console

As a cluster administrator, you can install the Performance Addon Operator using the web console.

**NOTE**

You must create the **Namespace** CR and **OperatorGroup** CR as mentioned in the previous section.

**Procedure**

1. Install the Performance Addon Operator using the OpenShift Container Platform web console:
  - a. In the OpenShift Container Platform web console, click **Operators** → **OperatorHub**.
  - b. Choose **Performance Addon Operator** from the list of available Operators, and then click **Install**.
  - c. On the **Install Operator** page, select **All namespaces on the cluster**. Then, click **Install**.
2. Optional: Verify that the performance-addon-operator installed successfully:
  - a. Switch to the **Operators** → **Installed Operators** page.
  - b. Ensure that **Performance Addon Operator** is listed in the **openshift-performance-addon-operator** 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 **performance-addon-operator** project.

## 14.3. UPGRADING PERFORMANCE ADDON OPERATOR

You can manually upgrade to the next minor version of Performance Addon Operator and monitor the status of an update by using the web console.

### 14.3.1. About upgrading Performance Addon Operator

- You can upgrade to the next minor version of Performance Addon Operator by using the OpenShift Container Platform web console to change the channel of your Operator subscription.
- You can enable automatic z-stream updates during Performance Addon Operator installation.
- Updates are delivered via the Marketplace Operator, which is deployed during OpenShift Container Platform installation. The Marketplace Operator makes external Operators available to your cluster.



- The amount of time an update takes to complete depends on your network connection. Most automatic updates complete within fifteen minutes.

#### 14.3.1.1. How Performance Addon Operator upgrades affect your cluster

- Neither the low latency tuning nor huge pages are affected.
- Updating the Operator should not cause any unexpected reboots.

#### 14.3.1.2. Upgrading Performance Addon Operator to the next minor version

You can manually upgrade Performance Addon Operator to the next minor version by using the OpenShift Container Platform web console to change the channel of your Operator subscription.

##### Prerequisites

- Access to the cluster as a user with the cluster-admin role.

##### Procedure

1. Access the OpenShift web console and navigate to **Operators → Installed Operators**.
2. Click **Performance Addon Operator** to open the **Operator Details** page.
3. Click the **Subscription** tab to open the **Subscription Overview** page.
4. In the **Channel** pane, click the pencil icon on the right side of the version number to open the **Change Subscription Update Channel** window.
5. Select the next minor version. For example, if you want to upgrade to Performance Addon Operator 4.7, select **4.7**.
6. Click **Save**.
7. Check the status of the upgrade by navigating to **Operators → Installed Operators**. You can also check the status by running the following **oc** command:

```
$ oc get csv -n openshift-performance-addon-operator
```

#### 14.3.1.3. Upgrading Performance Addon Operator when previously installed to a specific namespace

If you previously installed the Performance Addon Operator to a specific namespace on the cluster, for example **openshift-performance-addon-operator**, modify the **OperatorGroup** object to remove the **targetNamespaces** entry before upgrading.

##### Prerequisites

- Install the OpenShift Container Platform CLI (**oc**).
- Log in to the OpenShift cluster as a user with cluster-admin privileges.

##### Procedure

1. Edit the Performance Addon Operator **OperatorGroup** CR and remove the **spec** element that contains the **targetNamespaces** entry by running the following command:

```
$ oc patch operatorgroup -n openshift-performance-addon-operator openshift-performance-addon-operator --type json -p '[{"op": "remove", "path": "/spec"}]'
```

2. Wait until the Operator Lifecycle Manager (OLM) processes the change.
3. Verify that the OperatorGroup CR change has been successfully applied. Check that the **OperatorGroup** CR **spec** element has been removed:

```
$ oc describe -n openshift-performance-addon-operator og openshift-performance-addon-operator
```

4. Proceed with the Performance Addon Operator upgrade.

### 14.3.2. Monitoring upgrade status

The best way to monitor Performance Addon Operator upgrade status is to watch the **ClusterServiceVersion** (CSV) **PHASE**. You can also monitor the CSV conditions in the web console or by running the **oc get csv** command.



#### NOTE

The **PHASE** and conditions values are approximations that are based on available information.

#### Prerequisites

- Access to the cluster as a user with the **cluster-admin** role.
- Install the OpenShift CLI (**oc**).

#### Procedure

1. Run the following command:

```
$ oc get csv
```

2. Review the output, checking the **PHASE** field. For example:

```
VERSION  REPLACES                                PHASE
4.7.0    performance-addon-operator.v4.6.0           Installing
4.6.0                                         Replacing
```

3. Run **get csv** again to verify the output:

```
# oc get csv
```

#### Example output

```
NAME                DISPLAY                VERSION  REPLACES
PHASE
```

performance-addon-operator.v4.7.0 Performance Addon Operator 4.7.0 performance-addon-operator.v4.6.0 Succeeded

## 14.4. 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 a Performance Addon Operator to implement automatic tuning to achieve low latency performance and consistent response time 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 (real-time), the CPUs that will be reserved for housekeeping, and the CPUs that are used for running the workloads.

### 14.4.1. Known limitations for real-time



#### NOTE

The RT kernel is only supported on worker nodes.

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.

### 14.4.2. Provisioning a worker with real-time capabilities

1. Install Performance Addon Operator to the cluster.
2. Optional: Add a node to the OpenShift Container Platform cluster. See [Setting BIOS parameters](#).
3. Optional: Create a new machine config pool for real-time nodes.
4. Add the node to the proper machine config pool, using node role labels.  
You must decide which nodes will be configured with real-time workloads. It could be all of the nodes in the cluster or a subset of the nodes. The Performance Addon Operator expects all of the nodes are part of a dedicated machine config pool. If you use all of the nodes, you just point

the Performance Addon 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. Specify a node selector in the **PerformanceProfile**, as shown here:

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

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

```
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.

### 14.4.3. 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 **4.18.0-211.rt5.23.el8.x86\_64**:

| NAME                         | STATUS   | ROLES  | AGE   | VERSION                             | INTERNAL-IP |
|------------------------------|----------|--|-------|-------------------------------------|-------------|
| EXTERNAL-IP                  | OS-IMAGE |  |       | KERNEL-VERSION                      |             |
| CONTAINER-RUNTIME            |          |  |       |                                     |             |
| cnf-worker-0.example.com     | Ready    | worker,worker-rt                               | 5d17h | v1.20.0                             |             |
| 128.66.135.107               | <none>   | Red Hat Enterprise Linux                       |       | CoreOS 46.82.202008252340-0 (Ootpa) |             |
| 4.18.0-211.rt5.23.el8.x86_64 |          | cri-o://1.20.0-90.rhaos4.6.git4a0ac05.el8-rc.1 |       |                                     |             |
| [...]                        |          |  |       |                                     |             |

### 14.4.4. 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](#).

### 14.4.5. 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.

#### 14.4.6. Optional: Disabling CPU load balancing for DPDK

Functionality to disable or enable CPU load balancing is implemented on the CRI-O level. The code under the CRI-O disables or enables CPU load balancing only when the following requirements are met.

- The pod must use the **performance-<profile-name>** runtime class. You can get the proper name by looking at the status of the performance profile, as shown here:

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
...
status:
...
runtimeClass: performance-manual
```

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

The Performance Addon 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.

#### 14.4.7. 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](#) .

#### 14.4.8. 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 Performance Addon Operator. For more information, see [Assigning pods to nodes](#) .

#### 14.4.9. Managing device interrupt processing for guaranteed pod isolated CPUs

The Performance Addon Operator manages host CPUs by dividing them into reserved CPUs for cluster and operating system housekeeping duties, and isolated CPUs for workloads. CPUs that are used for low latency workloads are set 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, Performance Addon 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.

##### 14.4.9.1. Disabling global device interrupts handling in Performance Addon Operator

To configure Performance Addon 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
  ...
```

#### 14.4.9.2. 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** and **cpu-quota.crio.io** pod annotations to **disable**. An example pod specification snippet that illustrates this is below:

```
apiVersion: performance.openshift.io/v2
kind: Pod
metadata:
  annotations:
    irq-load-balancing.crio.io: "disable"
    cpu-quota.crio.io: "disable"
spec:
  runtimeClassName: performance-<profile_name>
  ...
```

#### 14.4.10. Upgrading the performance profile to use device interrupt processing

When you upgrade the Performance Addon Operator performance profile custom resource definition (CRD) from v1 or v1alpha1 to v2, **globallyDisableIrqLoadBalancing** is set to **true** on existing profiles.



#### NOTE

When **globallyDisableIrqLoadBalancing** is set to **true**, device interrupts are processed across all CPUs as long as they don't belong to a guaranteed pod.

#### 14.4.10.1. Supported API Versions

The Performance Addon 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**.

##### 14.4.10.1.1. Upgrading Performance Addon Operator API from v1alpha1 to v1

When upgrading Performance Addon 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 Performance Addon Operator with API version v1.

##### 14.4.10.1.2. Upgrading Performance Addon Operator API from v1alpha1 or v1 to v2

When upgrading from an older Performance Addon 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**.

#### 14.4.11. Configuring a node for IRQ dynamic load balancing

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



1. Log in to the OpenShift Container Platform cluster as a user with cluster-admin privileges.
2. Set the performance profile **apiVersion** to use **performance.openshift.io/v2**.
3. Remove the **globallyDisableIrqLoadBalancing** field or set it to **false**.
4. Set the appropriate isolated and reserved CPUs. The following snippet illustrates a profile that reserves 2 CPUs. IRQ load-balancing is enabled for pods running on the **isolated** CPU set:

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: dynamic-irq-profile
spec:
  cpu:
    isolated: 2-5
    reserved: 0-1
  ...
```

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: "quay.io/openshift-kni/cnf-tests:4.6"
    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>
```

- 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
```

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

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

### Expected output

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

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

sh-4.4#
```

- Verify that you can use the node file system:

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

### Expected output

```
sh-4.4#
```

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

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

### Example output

```
33
```

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

```
find /proc/irq/ -name smp_affinity_list -exec sh -c 'i="$1"; mask=$(cat $i); file=$(echo $i); echo $file: $mask' _ {} \;
```

### Example output

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

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

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

### 14.4.12. 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 could set logical cores CPU0 and CPU4 as isolated, and logical cores CPU1 and CPU5 as reserved:

```
...
cpu:
  isolated: 0-4
  reserved: 1-5
...
```



### 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.

#### 14.4.12.1. 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
```

## 14.5. CONFIGURING HUGE PAGES

Nodes must pre-allocate huge pages used in an OpenShift Container Platform cluster. Use the Performance Addon Operator to allocate huge pages on a specific node.

OpenShift Container Platform provides a method for creating and allocating huge pages. Performance Addon 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-###: ###
```

## 14.6. 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 Performance Addon 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
```

## 14.7. 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.

### Procedure

1. Prepare a cluster.
2. Create a machine config pool.
3. Install the Performance Addon Operator.
4. Create a performance profile that is appropriate for your hardware and topology. In the performance profile, you can specify whether to update the kernel to kernel-rt, allocation of huge pages, the CPUs that will be reserved for operating system housekeeping processes and CPUs that will be used for running the workloads.

This is a typical performance profile:

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: performance
spec:
  cpu:
    isolated: "5-15"
```

```

reserved: "0-4"
hugepages:
defaultHugepagesSize: "1G"
pages:
-size: "1G"
count: 16
node: 0
realTimeKernel:
enabled: true 1
numa: 2
topologyPolicy: "best-effort"
nodeSelector:
node-role.kubernetes.io/worker-cnf: ""

```

- 1 Valid values are **true** or **false**. Setting the **true** value installs the real-time kernel on the node.
- 2 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](#).

### 14.7.1. Partitioning the CPUs

You can reserve cores, or threads, for operating system housekeeping tasks from a single NUMA node and put your workloads on another NUMA node. The reason for this is that the housekeeping processes might be using the CPUs in a way that would impact latency sensitive processes running on those same CPUs. Keeping your workloads on a separate NUMA node prevents the processes from interfering with each other. Additionally, each NUMA node has its own memory bus that is not shared.

Specify two groups of CPUs in the **spec** section:

- **isolated** - Has the lowest latency. Processes in this group have no interruptions and so can, for example, reach much higher DPDK zero packet loss bandwidth.
- **reserved** - The housekeeping CPUs. Threads in the reserved group tend to be very busy, so latency-sensitive applications should be run in the isolated group. See [Create a pod that gets assigned a QoS class of Guaranteed](#).

## 14.8. PERFORMING END-TO-END TESTS FOR PLATFORM VERIFICATION

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

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

- Simple cluster validation
- Setup
- End to end tests

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



Validations include:

- Targeting a machine config pool that belong to the machines to be tested
- Enabling SCTP on the nodes
- Enabling xt\_u32 kernel module via machine config
- Having the Performance Addon Operator installed
- Having the SR-IOV Operator installed
- Having the PTP Operator installed
- Using OVN kubernetes as the SDN

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

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

### 14.8.1. Prerequisites

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

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

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

You can use the **ROLE\_WORKER\_CNF** variable to override the worker pool name:

```
$ docker run -v $(pwd)/:/kubeconfig -e KUBECONFIG=/kubeconfig/kubeconfig -e
ROLE_WORKER_CNF=custom-worker-pool registry.redhat.io/openshift4/cnf-tests-rhel8:v4.6
/usr/bin/test-run.sh
```



#### NOTE

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

## 14.8.2. Running the tests

Assuming the **kubeconfig** file is in the current folder, the command for running the test suite is:

```
$ docker run -v $(pwd)/:/kubeconfig -e KUBECONFIG=/kubeconfig/kubeconfig
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.6 /usr/bin/test-run.sh
```

This allows your **kubeconfig** file to be consumed from inside the running container.

### 14.8.2.1. Running the latency tests

In OpenShift Container Platform 4.7, you can also run latency tests from the CNF-test container. The latency test allows you to set a latency limit so that you can determine performance, throughput, and latency.

The latency test runs the **oslat** tool, which is an open source program to detect OS level latency. For more information, see the Red Hat Knowledgebase solution [How to measure OS and hardware latency on isolated CPUs?](#)

By default, the latency tests are disabled. To enable the latency test, you must add the **LATENCY\_TEST\_RUN** variable and set its value to **true**. For example, **LATENCY\_TEST\_RUN=true**.

Additionally, you can set the following environment variables for latency tests:

- **LATENCY\_TEST\_RUNTIME** - Specifies the amount of time (in seconds) that the latency test must run.
- **OSLAT\_MAXIMUM\_LATENCY** - Specifies the maximum latency (in microseconds) that is expected from all buckets during the **oslat** test run.

To perform the latency tests, run the following command:

```
$ docker run -v $(pwd)/:/kubeconfig -e KUBECONFIG=/kubeconfig/kubeconfig -e
LATENCY_TEST_RUN=true -e LATENCY_TEST_RUNTIME=600 -e
OSLAT_MAXIMUM_LATENCY=20 registry.redhat.io/openshift4/cnf-tests-rhel8:v4.7 /usr/bin/test-
run.sh
```



#### NOTE

You must run the latency test in discovery mode. For more information, see the Discovery mode section.

**Excerpt of a sample result of a 10-second latency test using the following command:**

```
[root@cnf12-installer ~]# podman run --rm -v $KUBECONFIG:/kubeconfig:Z -e
PERF_TEST_PROFILE=worker-cnf-2 -e KUBECONFIG=/kubeconfig -e LATENCY_TEST_RUN=true
-e LATENCY_TEST_RUNTIME=10 -e OSLAT_MAXIMUM_LATENCY=20 -e
DISCOVERY_MODE=true registry.redhat.io/openshift4/cnf-tests-rhel8:v4.7 /usr/bin/test-run.sh
-ginkgo.focus="Latency"
running /0_config.test -ginkgo.focus=Latency
```

## Example output

```
I1106 15:09:08.087085 7 request.go:621] Throttling request took 1.037172581s, request:
GET:https://api.cnf12.kni.lab.eng.bos.redhat.com:6443/apis/autoscaling.openshift.io/v1?timeout=32s
Running Suite: Performance Addon Operator configuration
```

```
Random Seed: 1604675347
Will run 0 of 1 specs
```

```
JUnit report was created: /unit_report_performance_config.xml
```

```
Ran 0 of 1 Specs in 0.000 seconds
SUCCESS! -- 0 Passed | 0 Failed | 0 Pending | 1 Skipped
PASS
```

```
running /4_latency.test -ginkgo.focus=Latency
```

```
I1106 15:09:10.735795 23 request.go:621] Throttling request took 1.037276624s, request:
GET:https://api.cnf12.kni.lab.eng.bos.redhat.com:6443/apis/certificates.k8s.io/v1?timeout=32s
Running Suite: Performance Addon Operator latency e2e tests
```

```
Random Seed: 1604675349
Will run 1 of 1 specs
```

```
I1106 15:10:06.401180 23 nodes.go:86] found mcd machine-config-daemon-r78qc for node
cnfdd8.clus2.t5g.lab.eng.bos.redhat.com
I1106 15:10:06.738120 23 utils.go:23] run command 'oc [exec -i -n openshift-machine-config-
operator -c machine-config-daemon --request-timeout 30 machine-config-daemon-r78qc -- cat
/rootfs/var/log/oslat.log]' (err=<nil>):
stdout=
Version: v0.1.7
```

```
Total runtime: 10 seconds
Thread priority: SCHED_FIFO:1
CPU list:
3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,3
8,39,40,41,42,43,44,45,46,47,48,49,50
CPU for main thread: 2
Workload: no
Workload mem: 0 (KiB)
Preheat cores: 48
```

```
Pre-heat for 1 seconds...
Test starts...
Test completed.
```

```
Core: 3 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
CPU Freq: 2096 2096 2096 2096 2096 2096 2096 2096 2096 2096 2096 2096 2096 2092 2096 2096
2096 2092 2092 2096 2096 2096 2096 2096 2096 2096 2096 2096 2092 2096 2096 2092 2096
```

```
2096 2096 2096 2092 2096 2096 2096 2092 2096 2096 2096 2096 2096 2096 (Mhz)
...
Maximum: 3 4 3 3 3 3 3 3 4 3 3 3 3 4 3 3 3 3 3 4 3 3 3 3 3 3 3 3 3 3 4 3 3 3 3 3 3 3 3 4 3 3 3 3 3 3 4 3 3 3 3 3 4 3 3 3 3 4
(us)
```

### 14.8.3. Image parameters

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

- **CNF\_TESTS\_IMAGE**
- **DPDK\_TESTS\_IMAGE**

For example, to change the **CNF\_TESTS\_IMAGE** with a custom registry run the following command:

```
$ docker run -v $(pwd)/:/kubeconfig -e KUBECONFIG=/kubeconfig/kubeconfig -e
CNF_TESTS_IMAGE="custom-cnf-tests-image:latest" registry.redhat.io/openshift4/cnf-tests-
rhel8:v4.7 /usr/bin/test-run.sh
```

#### 14.8.3.1. Ginkgo parameters

The test suite is built upon the ginkgo BDD framework. This means that it accepts parameters for filtering or skipping tests.

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

```
$ docker run -v $(pwd)/:/kubeconfig -e KUBECONFIG=/kubeconfig/kubeconfig
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.7 /usr/bin/test-run.sh -ginkgo.focus="performance|sctp"
```

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

To run only the latency test, you must provide the **-ginkgo.focus** parameter and the **PERF\_TEST\_PROFILE** environment variable that contains the name of the performance profile that needs to be tested. For example:

```
$ docker run --rm -v $KUBECONFIG:/kubeconfig -e KUBECONFIG=/kubeconfig -e
LATENCY_TEST_RUN=true -e LATENCY_TEST_RUNTIME=600 -e
OSLAT_MAXIMUM_LATENCY=20 -e PERF_TEST_PROFILE=<performance_profile_name>
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.7 /usr/bin/test-run.sh -ginkgo.focus="\[performance\]\
[config\]\[performance\]\ Latency\ Test"
```



#### NOTE

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

#### 14.8.3.2. Available features

The set of available features to filter are:

- **performance**
- **sriov**
- **ptp**
- **sctp**
- **xt\_u32**
- **dpdk**

#### 14.8.4. Dry run

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

```
$ docker run -v $(pwd)/:/kubeconfig -e KUBECONFIG=/kubeconfig/kubeconfig
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.7 /usr/bin/test-run.sh -ginkgo.dryRun -ginkgo.v
```

#### 14.8.5. Disconnected mode

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

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

##### 14.8.5.1. Mirroring the images to a custom registry accessible from the cluster

A **mirror** executable is shipped in the image to provide the input required by **oc** to mirror the images needed to run the tests to a local registry.

Run this command from an intermediate machine that has access both to the cluster and to [registry.redhat.io](https://registry.redhat.io) over the Internet:

```
$ docker run -v $(pwd)/:/kubeconfig -e KUBECONFIG=/kubeconfig/kubeconfig
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.7 /usr/bin/mirror -registry my.local.registry:5000/ | oc
image mirror -f -
```

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

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

This is done by setting the **IMAGE\_REGISTRY** environment variable:

```
$ docker run -v $(pwd)/:/kubeconfig -e KUBECONFIG=/kubeconfig/kubeconfig -e
IMAGE_REGISTRY="my.local.registry:5000/" -e CNF_TESTS_IMAGE="custom-cnf-tests-
image:latests" registry.redhat.io/openshift4/cnf-tests-rhel8:v4.7 /usr/bin/test-run.sh
```

##### 14.8.5.3. Mirroring to the cluster internal registry

OpenShift Container Platform provides a built-in container image registry, which runs as a standard workload on the cluster.

## Procedure

1. Gain external access to the registry by exposing it with a route:

```
$ oc patch configs.imageregistry.operator.openshift.io/cluster --patch '{"spec": {"defaultRoute":true}}' --type=merge
```

2. Fetch the registry endpoint:

```
REGISTRY=$(oc get route default-route -n openshift-image-registry --template='{{ .spec.host }}')
```

3. Create a namespace for exposing the images:

```
$ oc create ns cnftests
```

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

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

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

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

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

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

5. Retrieve the docker secret name and auth token:

```
SECRET=$(oc -n cnftests get secret | grep builder-docker | awk {'print $1'})
TOKEN=$(oc -n cnftests get secret $SECRET -o jsonpath="{.data[\".dockercfg\"]}" | base64 --decode | jq '.["image-registry.openshift-image-registry.svc:5000"].auth')
```

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

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

7. Do the mirroring:

```
$ docker run -v $(pwd)/:/kubeconfig -e KUBECONFIG=/kubeconfig/kubeconfig
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.6 /usr/bin/mirror -registry $REGISTRY/cnftests
| oc image mirror --insecure=true -a=$(pwd)/dockerauth.json -f -
```

8. Run the tests:

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

#### 14.8.5.4. Mirroring a different set of images

##### Procedure

1. The **mirror** command tries to mirror the u/s images by default. This can be overridden by passing a file with the following format to the image:

```
[
  {
    "registry": "public.registry.io:5000",
    "image": "imageforcnftests:4.7"
  },
  {
    "registry": "public.registry.io:5000",
    "image": "imagefordpdk:4.7"
  }
]
```

2. Pass it to the **mirror** command, for example saving it locally as **images.json**. With the following command, the local path is mounted in **/kubeconfig** inside the container and that can be passed to the mirror command.

```
$ docker run -v $(pwd)/:/kubeconfig -e KUBECONFIG=/kubeconfig/kubeconfig
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.6 /usr/bin/mirror --registry
"my.local.registry:5000/" --images "/kubeconfig/images.json" | oc image mirror -f -
```

#### 14.8.6. Discovery mode

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

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

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

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

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

### 14.8.6.1. Required environment configuration prerequisites

#### SR-IOV tests

Most SR-IOV tests require the following resources:

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

Some tests have additional requirements:

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

#### DPDK tests

The DPDK related tests require:

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

#### PTP tests

- A slave **PtpConfig** (**ptp4IOpts="-s"** ,**phc2sysOpts="-a -r"**).
- A node with a label matching the slave **PtpConfig**.

#### SCTP tests

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

#### XT\_U32 tests

- A node with a machine config that enables XT\_U32.

#### Performance Operator tests

Various tests have different requirements. Some of them are:

- A performance profile.



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

### 14.8.6.2. Limiting the nodes used during tests

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

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

### 14.8.6.3. Using a single performance profile

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

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

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

To override the performance profile used, the manifest must be mounted inside the container and the tests must be instructed by setting the **PERFORMANCE\_PROFILE\_MANIFEST\_OVERRIDE** parameter as follows:

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

### 14.8.6.4. Disabling the performance profile cleanup

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

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

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

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

### 14.8.7. Troubleshooting

The cluster must be reached from within the container. You can verify this by running:

```
$ docker run -v $(pwd)/:kubecfg -e KUBECFG=/kubecfg/kubecfg  
registry.redhat.io/openshift-kni/cnf-tests oc get nodes
```

If this does not work, it could be caused by spanning across DNS, MTU size, or firewall issues.

### 14.8.8. Test reports

CNF end-to-end tests produce two outputs: a JUnit test output and a test failure report.

#### 14.8.8.1. JUnit test output

A JUnit-compliant XML is produced by passing the **--junit** parameter together with the path where the report is dumped:

```
$ docker run -v $(pwd)/:kubecfg -v $(pwd)/junitdest:/path/to/junit -e  
KUBECFG=/kubecfg/kubecfg registry.redhat.io/openshift4/cnf-tests-rhel8:v4.6 /usr/bin/test-  
run.sh --junit /path/to/junit
```

#### 14.8.8.2. Test failure report

A report with information about the cluster state and resources for troubleshooting can be produced by passing the **--report** parameter with the path where the report is dumped:

```
$ docker run -v $(pwd)/:kubecfg -v $(pwd)/reportdest:/path/to/report -e  
KUBECFG=/kubecfg/kubecfg registry.redhat.io/openshift4/cnf-tests-rhel8:v4.6 /usr/bin/test-  
run.sh --report /path/to/report
```

#### 14.8.8.3. A note on podman

When executing podman as non root and non privileged, mounting paths can fail with "permission denied" errors. To make it work, append **:Z** to the volumes creation; for example, **-v \$(pwd)/:kubecfg:Z** to allow podman to do the proper SELinux relabeling.

#### 14.8.8.4. Running on OpenShift Container Platform 4.4

With the exception of the following, the CNF end-to-end tests are compatible with OpenShift Container Platform 4.4:

```
[test_id:28466][crit:high][vendor:cnf-qe@redhat.com][level:acceptance] Should contain configuration injected through openshift-node-performance profile
[test_id:28467][crit:high][vendor:cnf-qe@redhat.com][level:acceptance] Should contain configuration injected through the openshift-node-performance profile
```

You can skip these tests by adding the **-ginkgo.skip** "28466|28467" parameter.

#### 14.8.8.5. Using a single performance profile

The DPDK tests require more resources than what is required by the performance test suite. To make the execution faster, you can override the performance profile used by the tests using a profile that also serves the DPDK test suite.

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

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

To override the performance profile, the manifest must be mounted inside the container and the tests must be instructed by setting the **PERFORMANCE\_PROFILE\_MANIFEST\_OVERRIDE**:

```
$ docker run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig -e
PERFORMANCE_PROFILE_MANIFEST_OVERRIDE=/kubeconfig/manifest.yaml
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.6 /usr/bin/test-run.sh
```

#### 14.8.9. Impacts on the cluster

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

##### 14.8.9.1. SCTP

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

#### 14.8.9.2. XT\_U32

XT\_U32 tests run pods on different nodes to check iptables rule that utilize xt\_u32. The impacts on the cluster are related to running simple pods on two nodes.

#### 14.8.9.3. SR-IOV

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

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

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

#### 14.8.9.4. PTP

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

#### 14.8.9.5. Performance

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

#### 14.8.9.6. DPDK

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

#### 14.8.9.7. Cleaning up

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

### 14.9. 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 Performance Addon 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.

**14.9.1. 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 addons that encompass kernel args, kube config, huge pages allocation, and deployment of rt-kernel. The performance addons 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
```

## 14.10. 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.

### 14.10.1. 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.

### 14.10.2. 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 Performance Addon 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 container-native virtualization data with **must-gather**, you must specify the container-native virtualization image:

```
--image=registry.redhat.io/openshift4/performance-addon-operator-must-gather-rhel8.
```

### 14.10.3. 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.

#### 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 container-native virtualization:

```
$ oc adm must-gather \
  --image-stream=openshift/must-gather \ 1
  --image=registry.redhat.io/openshift4/performance-addon-operator-must-gather-rhel8 2
```

1 The default OpenShift Container Platform **must-gather** image.

2 The **must-gather** image for low latency tuning diagnostics.

3. Create a compressed file from the **must-gather** directory that was created in your working directory. For example, on a computer that uses a Linux operating system, run the following command:

```
$ tar cvaf must-gather.tar.gz must-gather.local.5421342344627712289/ 1
```

1 Replace **must-gather-local.5421342344627712289/** with the actual directory name.

4. Attach the compressed file to your support case on the [Red Hat Customer Portal](#).

#### Additional resources

- For more information about MachineConfig and KubeletConfig, see [Managing nodes](#).
- For more information about the Node Tuning Operator, see [Using the Node Tuning Operator](#).
- For more information about the PerformanceProfile, see [Configuring huge pages](#).



- For more information about consuming huge pages from your containers, see [How huge pages are consumed by apps](#).