



# OpenShift Container Platform 4.6

## Scalability and performance

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



# OpenShift Container Platform 4.6 Scalability and performance

---

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

## Legal Notice

Copyright © 2022 Red Hat, Inc.

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

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

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

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

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

Linux<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	7
2.3. CONTROL PLANE NODE SIZING	10
2.3.1. Increasing the flavor size of the Amazon Web Services (AWS) master instances	11
2.4. RECOMMENDED ETCD PRACTICES	12
2.5. DEFRAGMENTING ETCD DATA	13
2.6. OPENSIFT CONTAINER PLATFORM INFRASTRUCTURE COMPONENTS	16
2.7. MOVING THE MONITORING SOLUTION	16
2.8. MOVING THE DEFAULT REGISTRY	17
2.9. MOVING THE ROUTER	19
2.10. INFRASTRUCTURE NODE SIZING	20
2.11. ADDITIONAL RESOURCES	21
<b>CHAPTER 3. RECOMMENDED CLUSTER SCALING PRACTICES</b> .....	<b>22</b>
3.1. RECOMMENDED PRACTICES FOR SCALING THE CLUSTER	22
3.2. MODIFYING A MACHINE SET	22
3.3. ABOUT MACHINE HEALTH CHECKS	23
3.3.1. MachineHealthChecks on Bare Metal	24
3.3.2. Limitations when deploying machine health checks	24
3.4. SAMPLE MACHINEHEALTHCHECK RESOURCE	25
3.4.1. Short-circuiting machine health check remediation	26
3.4.1.1. Setting maxUnhealthy by using an absolute value	27
3.4.1.2. Setting maxUnhealthy by using percentages	27
3.5. CREATING A MACHINEHEALTHCHECK RESOURCE	27
<b>CHAPTER 4. USING THE NODE TUNING OPERATOR</b> .....	<b>29</b>
4.1. ABOUT THE NODE TUNING OPERATOR	29
4.2. ACCESSING AN EXAMPLE NODE TUNING OPERATOR SPECIFICATION	29
4.3. DEFAULT PROFILES SET ON A CLUSTER	30
4.4. VERIFYING THAT THE TUNED PROFILES ARE APPLIED	31
4.5. CUSTOM TUNING SPECIFICATION	32
4.6. CUSTOM TUNING EXAMPLE	36
4.7. SUPPORTED TUNED DAEMON PLUG-INS	37
<b>CHAPTER 5. USING CLUSTER LOADER</b> .....	<b>39</b>
5.1. INSTALLING CLUSTER LOADER	39
5.2. RUNNING CLUSTER LOADER	39
5.3. CONFIGURING CLUSTER LOADER	39
5.3.1. Example Cluster Loader configuration file	40
5.3.2. Configuration fields	41
5.4. KNOWN ISSUES	44
<b>CHAPTER 6. USING CPU MANAGER</b> .....	<b>45</b>
6.1. SETTING UP CPU MANAGER	45
<b>CHAPTER 7. USING TOPOLOGY MANAGER</b> .....	<b>50</b>
7.1. TOPOLOGY MANAGER POLICIES	50
7.2. SETTING UP TOPOLOGY MANAGER	51
7.3. POD INTERACTIONS WITH TOPOLOGY MANAGER POLICIES	51

<b>CHAPTER 8. SCALING THE CLUSTER MONITORING OPERATOR</b> .....	<b>53</b>
8.1. PROMETHEUS DATABASE STORAGE REQUIREMENTS	53
8.2. CONFIGURING CLUSTER MONITORING	54
<b>CHAPTER 9. PLANNING YOUR ENVIRONMENT ACCORDING TO OBJECT MAXIMUMS</b> .....	<b>56</b>
9.1. OPENSIFT CONTAINER PLATFORM TESTED CLUSTER MAXIMUMS FOR MAJOR RELEASES	56
9.2. OPENSIFT CONTAINER PLATFORM ENVIRONMENT AND CONFIGURATION ON WHICH THE CLUSTER MAXIMUMS ARE TESTED	58
9.3. HOW TO PLAN YOUR ENVIRONMENT ACCORDING TO TESTED CLUSTER MAXIMUMS	58
9.4. HOW TO PLAN YOUR ENVIRONMENT ACCORDING TO APPLICATION REQUIREMENTS	59
<b>CHAPTER 10. OPTIMIZING STORAGE</b> .....	<b>63</b>
10.1. AVAILABLE PERSISTENT STORAGE OPTIONS	63
10.2. RECOMMENDED CONFIGURABLE STORAGE TECHNOLOGY	64
10.2.1. Specific application storage recommendations	64
10.2.1.1. Registry	65
10.2.1.2. Scaled registry	65
10.2.1.3. Metrics	65
10.2.1.4. Logging	66
10.2.1.5. Applications	66
10.2.2. Other specific application storage recommendations	66
10.3. DATA STORAGE MANAGEMENT	66
<b>CHAPTER 11. OPTIMIZING ROUTING</b> .....	<b>68</b>
11.1. BASELINE INGRESS CONTROLLER (ROUTER) PERFORMANCE	68
11.2. INGRESS CONTROLLER (ROUTER) PERFORMANCE OPTIMIZATIONS	69
<b>CHAPTER 12. OPTIMIZING NETWORKING</b> .....	<b>70</b>
12.1. OPTIMIZING THE MTU FOR YOUR NETWORK	70
12.2. RECOMMENDED PRACTICES FOR INSTALLING LARGE SCALE CLUSTERS	71
12.3. IMPACT OF IPSEC	71
<b>CHAPTER 13. MANAGING BARE METAL HOSTS</b> .....	<b>72</b>
13.1. ABOUT BARE METAL HOSTS AND NODES	72
13.2. MAINTAINING BARE METAL HOSTS	72
13.2.1. Adding a bare metal host to the cluster using the web console	72
13.2.2. Adding a bare metal host to the cluster using YAML in the web console	73
13.2.3. Automatically scaling machines to the number of available bare metal hosts	74
<b>CHAPTER 14. WHAT HUGE PAGES DO AND HOW THEY ARE CONSUMED BY APPLICATIONS</b> .....	<b>76</b>
14.1. WHAT HUGE PAGES DO	76
14.2. HOW HUGE PAGES ARE CONSUMED BY APPS	76
14.3. CONFIGURING HUGE PAGES	77
14.3.1. At boot time	77
<b>CHAPTER 15. PERFORMANCE ADDON OPERATOR FOR LOW LATENCY NODES</b> .....	<b>80</b>
15.1. UNDERSTANDING LOW LATENCY	80
15.2. INSTALLING THE PERFORMANCE ADDON OPERATOR	80
15.2.1. Installing the Operator using the CLI	80
15.2.2. Installing the Performance Addon Operator using the web console	82
15.3. UPGRADING PERFORMANCE ADDON OPERATOR	83
15.3.1. About upgrading Performance Addon Operator	83
15.3.1.1. How Performance Addon Operator upgrades affect your cluster	83
15.3.1.2. Upgrading Performance Addon Operator to the next minor version	83
15.3.2. Monitoring upgrade status	84

15.4. PROVISIONING REAL-TIME AND LOW LATENCY WORKLOADS	85
15.4.1. Known limitations for real-time	85
15.4.2. Provisioning a worker with real-time capabilities	85
15.4.3. Verifying the real-time kernel installation	87
15.4.4. Creating a workload that works in real-time	87
15.4.5. Creating a pod with a QoS class of Guaranteed	87
15.4.6. Optional: Disabling CPU load balancing for DPDK	89
15.4.7. Assigning a proper node selector	89
15.4.8. Scheduling a workload onto a worker with real-time capabilities	90
15.5. CONFIGURING HUGE PAGES	90
15.6. ALLOCATING MULTIPLE HUGE PAGE SIZES	91
15.7. RESTRICTING CPUS FOR INFRA AND APPLICATION CONTAINERS	91
15.8. TUNING NODES FOR LOW LATENCY WITH THE PERFORMANCE PROFILE	93
15.9. PERFORMING END-TO-END TESTS FOR PLATFORM VERIFICATION	94
15.9.1. Prerequisites	95
15.9.2. Running the tests	96
15.9.3. Image parameters	96
15.9.3.1. Ginkgo parameters	96
15.9.3.2. Available features	96
15.9.4. Dry run	97
15.9.5. Disconnected mode	97
15.9.5.1. Mirroring the images to a custom registry accessible from the cluster	97
15.9.5.2. Instruct the tests to consume those images from a custom registry	97
15.9.5.3. Mirroring to the cluster internal registry	97
15.9.5.4. Mirroring a different set of images	99
15.9.6. Discovery mode	99
15.9.6.1. Required environment configuration prerequisites	99
15.9.6.2. Limiting the nodes used during tests	100
15.9.6.3. Using a single performance profile	101
15.9.6.4. Disabling the performance profile cleanup	101
15.9.7. Troubleshooting	102
15.9.8. Test reports	102
15.9.8.1. JUnit test output	102
15.9.8.2. Test failure report	102
15.9.8.3. A note on podman	102
15.9.8.4. Running on OpenShift Container Platform 4.4	102
15.9.8.5. Using a single performance profile	103
15.9.9. Impacts on the cluster	103
15.9.9.1. SCTP	103
15.9.9.2. SR-IOV	103
15.9.9.3. PTP	104
15.9.9.4. Performance	104
15.9.9.5. DPDK	104
15.9.9.6. Cleaning up	104
15.10. DEBUGGING LOW LATENCY CNF TUNING STATUS	104
15.10.1. Machine config pools	105
15.11. COLLECTING LOW LATENCY TUNING DEBUGGING DATA FOR RED HAT SUPPORT	106
15.11.1. About the must-gather tool	107
15.11.2. About collecting low latency tuning data	107
15.11.3. Gathering data about specific features	107
<b>CHAPTER 16. OPTIMIZING DATA PLANE PERFORMANCE WITH THE INTEL FPGA PAC N3000 AND INTEL VRAN DEDICATED ACCELERATOR ACC100</b>	<b>109</b>

16.1. UNDERSTANDING INTEL HARDWARE ACCELERATOR CARDS FOR OPENSIFT CONTAINER PLATFORM	109
Intel FPGA PAC N3000	109
vRAN Dedicated Accelerator ACC100	109
16.2. INSTALLING THE OPENNESS OPERATOR FOR INTEL FPGA PAC N3000	109
16.2.1. Installing the Operator by using the CLI	110
16.2.2. Installing the OpenNESS Operator for Intel FPGA PAC N3000 Operator by using the web console	111
16.3. PROGRAMMING THE OPENNESS OPERATOR FOR INTEL FPGA PAC N3000	112
16.3.1. Programming the N3000 with a vRAN bitstream	112
16.4. INSTALLING THE OPENNESS SR-IOV OPERATOR FOR WIRELESS FEC ACCELERATORS	118
16.4.1. Installing the OpenNESS SR-IOV Operator for Wireless FEC Accelerators by using the CLI	119
16.4.2. Installing the OpenNESS SR-IOV Operator for Wireless FEC Accelerators by using the web console	121
16.4.3. Configuring the SR-IOV-FEC Operator for Intel FPGA PAC N3000	122
16.4.4. Configuring the SR-IOV-FEC Operator for the Intel vRAN Dedicated Accelerator ACC100	129
16.4.5. Verifying application pod access and FPGA usage on OpenNESS	136
16.5. ADDITIONAL RESOURCES	140





# 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.



### IMPORTANT

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

## 2.1. RECOMMENDED NODE HOST PRACTICES

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

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

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



### IMPORTANT

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

**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.



## NOTE

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

## Procedure

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

```
$ oc get machineconfig
```

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

2. 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. Update the machine config pool for workers with the label:

```
$ oc label machineconfigpool worker custom-kubelet=large-pods
```

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

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

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

```
$ 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

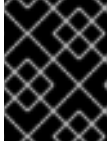
By default, only one machine is allowed to be unavailable when applying the kubelet-related configuration to the available worker nodes. For a large cluster, it can take a long time for the configuration change to be reflected. At any time, you can adjust the number of machines that are updating to speed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

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

```
$ oc edit machineconfigpool worker
```

- 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. The control plane tests create the following objects across the cluster in each of the namespaces depending on the node counts:

- 12 image streams
- 3 build configurations
- 6 builds
- 1 deployment with 2 pod replicas mounting two secrets each
- 2 deployments with 1 pod replica mounting two secrets
- 3 services pointing to the previous deployments
- 3 routes pointing to the previous deployments
- 10 secrets, 2 of which are mounted by the previous deployments
- 10 config maps, 2 of which are mounted by the previous deployments

Number of worker nodes	Cluster load (namespaces)	CPU cores	Memory (GB)
25	500	4	16
100	1000	8	32
250	4000	16	96

On a large and dense cluster with three masters or control plane nodes, the CPU and memory usage will spike up when one of the nodes is stopped, rebooted or fails. The failures can be due to unexpected issues with power, network or underlying infrastructure in addition to intentional cases where the cluster is restarted after shutting it down to save costs. The remaining two control plane nodes must handle the load in order to be highly available which leads to increase in the resource usage. This is also expected during upgrades because the masters are cordoned, drained, and rebooted serially to apply the

operating system updates, as well as the control plane Operators update. To avoid cascading failures, keep the overall CPU and memory resource usage on the control plane nodes to at most 60% of all available capacity to handle the resource usage spikes. Increase the CPU and memory on the control plane nodes accordingly to avoid potential downtime due to lack of resources.



### IMPORTANT

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



### IMPORTANT

If you used an installer-provisioned infrastructure installation method, you cannot modify the control plane node size in a running OpenShift Container Platform 4.6 cluster. Instead, you must estimate your total node count and use the suggested control plane node size during installation.



### IMPORTANT

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



### NOTE

In OpenShift Container Platform 4.6, 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.

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

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



### NOTE

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

### Prerequisites

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

### Procedure

1. Open the AWS console, fetch the master instances.
2. Stop one master instance.
3. Select the stopped instance, and click **Actions** → **Instance Settings** → **Change instance type**.

4. Change the instance to a larger type, ensuring that the type is the same base as the previous selection, and apply changes. For example, you can change **m5.xlarge** to **m5.2xlarge** or **m5.4xlarge**.
5. Backup the instance, and repeat the steps for the next master instance.

### Additional resources

- [Backing up etcd](#)

## 2.4. RECOMMENDED ETCD PRACTICES

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

### Monitor these key metrics:

- **etcd\_server\_quota\_backend\_bytes**, which is the current quota limit
- **etcd\_mvcc\_db\_total\_size\_in\_use\_in\_bytes**, which indicates the actual database usage after a history compaction
- **etcd\_debugging\_mvcc\_db\_total\_size\_in\_bytes**, which shows the database size, including free space waiting for defragmentation

For more information about defragmenting etcd, see the "Defragmenting etcd data" section.

Because etcd writes data to disk and persists proposals on disk, its performance depends on disk performance. Slow disks and disk activity from other processes can cause long fsync latencies. Those latencies can cause etcd to miss heartbeats, not commit new proposals to the disk on time, and ultimately experience request timeouts and temporary leader loss. Run etcd on machines that are backed by SSD or NVMe disks with low latency and high throughput. Consider single-level cell (SLC) solid-state drives (SSDs), which provide 1 bit per memory cell, are durable and reliable, and are ideal for write-intensive workloads.

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

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

To validate the hardware for etcd before or after you create the OpenShift Container Platform cluster, you can use an I/O benchmarking tool called fio.

### Prerequisites

- Container runtimes such as Podman or Docker are installed on the machine that you're testing.



- Data is written to the **/var/lib/etcd** path.

### Procedure

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

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

- If you use Docker, run this command:

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

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

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

The **histogram\_quantile(0.99, rate(etcd\_network\_peer\_round\_trip\_time\_seconds\_bucket[2m]))** metric reports the round trip time for etcd to finish replicating the client requests between the members. Ensure that it is less than 50 ms.

## 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



- 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 HAProxy-based Ingress Controller
- The cluster metrics collection, or monitoring service, including components for monitoring user-defined projects
- Cluster aggregated logging
- Service brokers
- Red Hat Quay
- Red Hat OpenShift Container Storage
- Red Hat Advanced Cluster Manager
- Red Hat Advanced Cluster Security for Kubernetes
- Red Hat OpenShift GitOps
- Red Hat OpenShift Pipelines

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

### Additional resources

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

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

```
spec:
  affinity:
    podAntiAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
      - podAffinityTerm:
          namespaces:
          - openshift-image-registry
```

```

topologyKey: kubernetes.io/hostname
weight: 100
logLevel: Normal
managementState: Managed
nodeSelector:
  node-role.kubernetes.io/infra: ""

```

4. Verify the registry pod has been moved to the infrastructure node.
  - a. Run the following command to identify the node where the registry pod is located:

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

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

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

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

## 2.9. 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
router-default-955d875f4-255g8  0/1    Terminating 0          19h  10.129.2.4    ip-10-0-148-172.ec2.internal
```

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

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

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

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

#### Example output

```
NAME                                STATUS  ROLES    AGE  VERSION
ip-10-0-217-226.ec2.internal  Ready  infra,worker  17h  v1.19.0
```

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.



### IMPORTANT

The sizing recommendations below are only applicable for the Prometheus, Router, and Registry infrastructure components, which are installed during cluster installation. Logging is a day-two operation and its sizing recommendation are given in the last two columns.

Number of worker nodes	CPU cores	Memory (GB)	CPU cores with Logging	Memory (GB) with Logging
25	4	16	4	64
100	8	32	8	128
250	16	128	16	128
500	32	128	32	192



### 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 4.6 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.



### NOTE

In OpenShift Container Platform 4.6, 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.

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

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

### 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

You can define conditions under which machines in a cluster are considered unhealthy by using a **MachineHealthCheck** resource. Machines matching the conditions are automatically remediated.

To monitor machine health, create a **MachineHealthCheck** custom resource (CR) that includes a label for the set of machines to monitor and a condition to check, such as staying in the **NotReady** status for 15 minutes or displaying a permanent condition in the node-problem-detector.

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



## NOTE

For machines with the master role, the machine health check reports the number of unhealthy nodes, but the machine is not deleted. For example:

### Example output

```
$ oc get machinehealthcheck example -n openshift-machine-api
```

NAME	MAXUNHEALTHY	EXPECTEDMACHINES	CURRENTHEALTHY
example	40%	3	1

To limit the disruptive impact of machine deletions, 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, the controller stops deleting machines and you must manually intervene.

To stop the check, remove the custom resource.

### 3.3.1. MachineHealthChecks on Bare Metal

Machine deletion on bare metal cluster triggers reprovisioning of a bare metal host. Usually bare metal reprovisioning is a lengthy process, during which the cluster is missing compute resources and applications might be interrupted. To change the default remediation process from machine deletion to host power-cycle, annotate the MachineHealthCheck resource with the **machine.openshift.io/remediation-strategy: external-baremetal** annotation.

After you set the annotation, unhealthy machines are power-cycled by using BMC credentials.

### 3.3.2. 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 resembles one of the following YAML files:

#### MachineHealthCheck for bare metal

```

apiVersion: machine.openshift.io/v1beta1
kind: MachineHealthCheck
metadata:
  name: example 1
  namespace: openshift-machine-api
  annotations:
    machine.openshift.io/remediation-strategy: external-baremetal 2
spec:
  selector:
    matchLabels:
      machine.openshift.io/cluster-api-machine-role: <role> 3
      machine.openshift.io/cluster-api-machine-type: <role> 4
      machine.openshift.io/cluster-api-machineset: <cluster_name>-<label>-<zone> 5
  unhealthyConditions:
  - type: "Ready"
    timeout: "300s" 6
    status: "False"
  - type: "Ready"
    timeout: "300s" 7
    status: "Unknown"
  maxUnhealthy: "40%" 8
  nodeStartupTimeout: "10m" 9

```

- 1 Specify the name of the machine health check to deploy.
- 2 For bare metal clusters, you must include the **machine.openshift.io/remediation-strategy: external-baremetal** annotation in the **annotations** section to enable power-cycle remediation. With this remediation strategy, unhealthy hosts are rebooted instead of removed from the cluster.
- 3 4 Specify a label for the machine pool that you want to check.
- 5 Specify the machine set to track in **<cluster\_name>-<label>-<zone>** format. For example, **prod-node-us-east-1a**.
- 6 7 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.
- 8 Specify the amount of machines allowed to be concurrently remediated in the targeted pool. This can be set as a percentage or an integer. If the number of unhealthy machines exceeds the limit set by **maxUnhealthy**, remediation is not performed.
- 9 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.

**MachineHealthCheck for all other installation types**

```

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

```

- 1** Specify the name of the machine health check to deploy.
- 2** **3** Specify a label for the machine pool that you want to check.
- 4** Specify the machine set to track in **<cluster\_name>-<label>-<zone>** format. For example, **prod-node-us-east-1a**.
- 5** **6** Specify the timeout duration for a node condition. If a condition is met for the duration of the timeout, the machine will be remediated. Long timeouts can result in long periods of downtime for a workload on an unhealthy machine.
- 7** Specify the amount of machines allowed to be concurrently remediated in the targeted pool. This can be set as a percentage or an integer. If the number of unhealthy machines exceeds the limit set by **maxUnhealthy**, remediation is not performed.
- 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 appropriate **maxUnhealthy** value depends on the scale of the cluster you deploy and how many machines the **MachineHealthCheck** covers. For example, you can use the **maxUnhealthy** value to cover multiple machine sets across multiple availability zones so that if you lose an entire zone, your **maxUnhealthy** setting prevents further remediation within the cluster.

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: ❶
  <mcLabels> ❷
  match: ❸
  <match> ❹
  priority: <priority> ❺
  profile: <tuned_profile_name> ❻
```

- 
- 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:

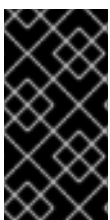
```
- label: <label_name> 1
  value: <label_value> 2
  type: <label_type> 3
  <match> 4
```

- 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. To target nodes that have both master and worker roles, you must use the master role.

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



### IMPORTANT

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

## Example: node or pod label based matching

```

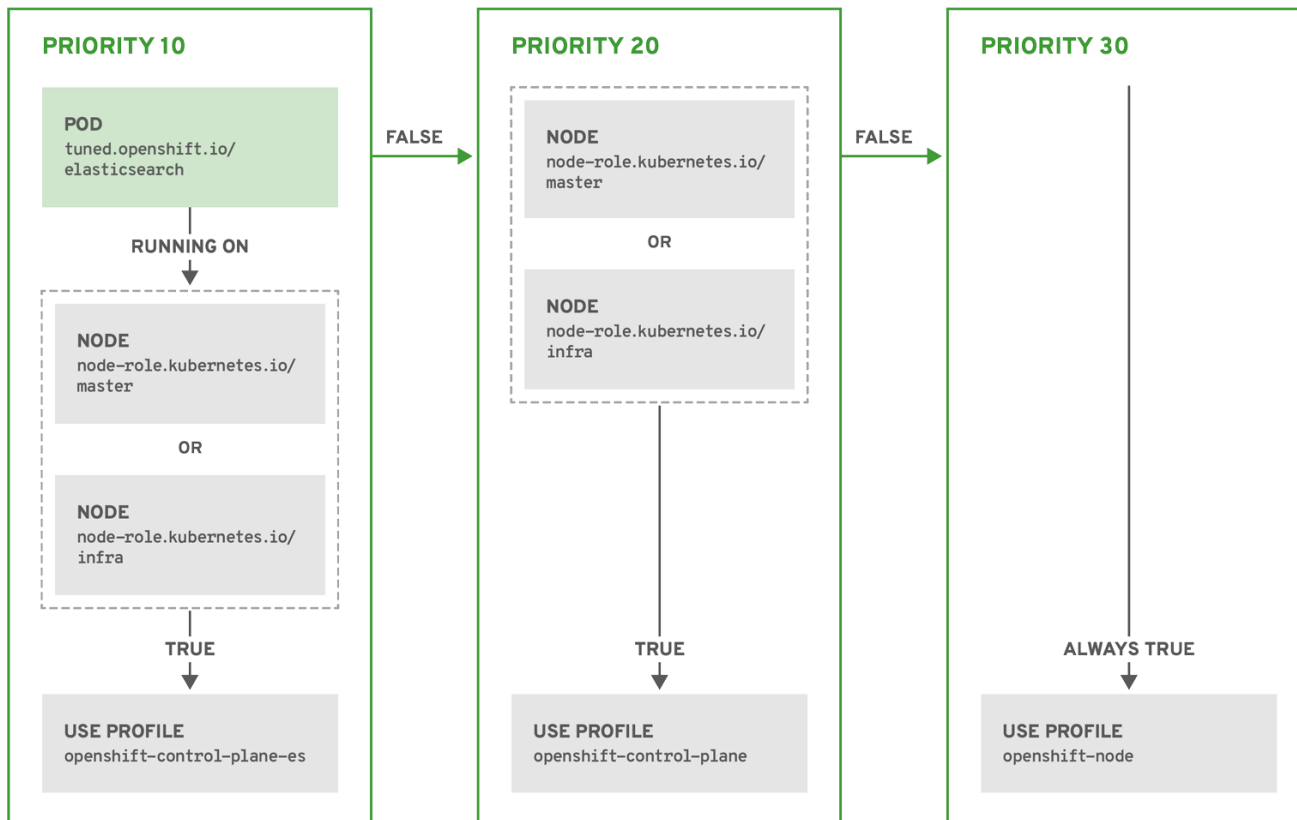
- match:
  - label: tuned.openshift.io/elasticsearch
    match:
      - label: node-role.kubernetes.io/master
      - label: node-role.kubernetes.io/infra
    type: pod
  priority: 10
  profile: openshift-control-plane-es
- match:
  - label: node-role.kubernetes.io/master
  - label: node-role.kubernetes.io/infra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

```

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

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

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



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:4.6
```

### 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:4.6 /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:4.6 \
/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

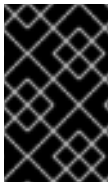
    - num: 1
      basename: clusterloader-nodejs-mongodb
      tuning: default
      ifexists: reuse
      templates:
        - num: 1
          file: quickstarts/nodejs-mongodb.json

    - 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.

### 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

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

### Procedure

To increase the storage capacity for Prometheus:

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

```

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

```

- 1 A typical value is **PROMETHEUS\_RETENTION\_PERIOD=15d**. Units are measured in time using one of these suffixes: s, m, h, d.
- 2 4 The storage class for your cluster.
- 3 A typical value is **PROMETHEUS\_STORAGE\_SIZE=2000Gi**. Storage values can be a plain integer or as a fixed-point integer using one of these suffixes: E, P, T, G, M, K. You can also use the power-of-two equivalents: Ei, Pi, Ti, Gi, Mi, Ki.
- 5 A typical value is **ALERTMANAGER\_STORAGE\_SIZE=20Gi**. Storage values can be a plain integer or as a fixed-point integer using one of these suffixes: E, P, T, G, M, K. You can also use the power-of-two equivalents: Ei, Pi, Ti, Gi, Mi, Ki.

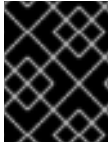
2. Add values for the retention period, storage class, and storage sizes.
3. Save the file.
4. Apply the changes by running:

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

## CHAPTER 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.



### IMPORTANT

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

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

### 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) - Pipeline Strategy	10,000 (Default pod RAM 512 Mi) - Source-to-Image (S2I) build strategy
Number of pods per namespace <sup>[4]</sup>	25,000	25,000
Number of routes and back ends per Ingress Controller	2,000 per router	2,000 per router
Number of secrets	80,000	80,000

Maximum type	3.x tested maximum	4.x tested maximum
Number of config maps	90,000	90,000
Number of services <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
Number of build configs	12,000	12,000
Number of secrets	40,000	40,000
Number of custom resource definitions (CRD)	There is no default value.	512 <sup>[6]</sup>

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 keyspace 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.
6. OpenShift Container Platform has a limit of 512 total custom resource definitions (CRD), including those installed by OpenShift Container Platform, products integrating with OpenShift Container Platform and user created CRDs. If there are more than 512 CRDs created, then there is a possibility that **oc** commands requests may be throttled.

**NOTE**

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

## 9.2. 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.

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



## IMPORTANT

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

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

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

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

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

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

### Example scenario

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

$$2200 / 500 = 4.4$$

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

$$2200 / 20 = 110$$

Where:

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

## 9.4. HOW TO PLAN YOUR ENVIRONMENT ACCORDING TO APPLICATION REQUIREMENTS

Consider an example application environment:

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

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

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

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

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

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

```
---
apiVersion: v1
kind: Template
metadata:
  name: deployment-config-template
  creationTimestamp:
  annotations:
    description: This template will create a deploymentConfig with 1 replica, 4 env vars and a service.
    tags: ""
objects:
- apiVersion: v1
  kind: DeploymentConfig
  metadata:
    name: deploymentconfig${IDENTIFIER}
  spec:
    template:
      metadata:
        labels:
          name: replicationcontroller${IDENTIFIER}
      spec:
        enableServiceLinks: false
        containers:
        - name: pause${IDENTIFIER}
```



```

image: "${IMAGE}"
ports:
- containerPort: 8080
  protocol: TCP
env:
- name: ENVVAR1_${IDENTIFIER}
  value: "${ENV_VALUE}"
- name: ENVVAR2_${IDENTIFIER}
  value: "${ENV_VALUE}"
- name: ENVVAR3_${IDENTIFIER}
  value: "${ENV_VALUE}"
- name: ENVVAR4_${IDENTIFIER}
  value: "${ENV_VALUE}"
resources: {}
imagePullPolicy: IfNotPresent
capabilities: {}
securityContext:
  capabilities: {}
  privileged: false
restartPolicy: Always
serviceAccount: "
replicas: 1
selector:
  name: replicationcontroller${IDENTIFIER}
triggers:
- type: ConfigChange
strategy:
  type: Rolling
- apiVersion: v1
kind: Service
metadata:
  name: service${IDENTIFIER}
spec:
  selector:
    name: replicationcontroller${IDENTIFIER}
  ports:
- name: serviceport${IDENTIFIER}
  protocol: TCP
  port: 80
  targetPort: 8080
  portName: ""
  type: ClusterIP
  sessionAffinity: None
status:
  loadBalancer: {}
parameters:
- name: IDENTIFIER
  description: Number to append to the name of resources
  value: '1'
  required: true
- name: IMAGE
  description: Image to use for deploymentConfig
  value: gcr.io/google-containers/pause-amd64:3.0
  required: false
- name: ENV_VALUE
  description: Value to use for environment variables

```

```
generate: expression
from: "[A-Za-z0-9]{255}"
required: false
labels:
  template: deployment-config-template
```

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

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

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

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

## CHAPTER 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.

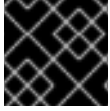
### 10.1. AVAILABLE PERSISTENT STORAGE OPTIONS

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 4.6.

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

### 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:

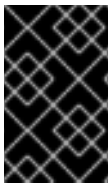
- The preferred storage technology is block storage.
- Object storage is not configurable.

### 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



### IMPORTANT

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

- Red Hat OpenStack Platform (RHOSP) Cinder: RHOSP Cinder tends to be adept in ROX access mode use cases.
- Databases: Databases (RDBMSs, NoSQL DBs, etc.) tend to perform best with dedicated block storage.
- The etcd database must have enough storage and adequate performance capacity to enable a large cluster. Information about monitoring and benchmarking tools to establish ample storage and a high-performance environment is described in *Recommended etcd practices*.

## 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
<code>/var/log</code>	Log files for all components.	10 to 30 GB.	Log files can grow quickly; size can be managed by growing disks or by using log rotate.

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

## 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 controllers (NIC) offloads, multi-queue, and ethtool settings.

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

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

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

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

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

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

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

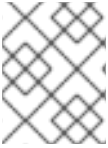
### 12.1. OPTIMIZING THE MTU FOR YOUR NETWORK

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

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

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

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

**NOTE**

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

## 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. MANAGING BARE METAL HOSTS

When you install OpenShift Container Platform on a bare metal cluster, you can provision and manage bare metal nodes using **machine** and **machineset** custom resources (CRs) for bare metal hosts that exist in the cluster.

### 13.1. ABOUT BARE METAL HOSTS AND NODES

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

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

### 13.2. MAINTAINING BARE METAL HOSTS

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

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

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

1. Annotates the bare metal host CR with **cluster.k8s.io/delete-machine: true**
2. Scales down the related machine set



#### NOTE

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

#### Additional resources

- [Adding compute machines to bare metal](#)

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

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

#### Prerequisites

- Install an RHCOS cluster on bare metal.

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

### Procedure

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



### NOTE

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

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

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

### Prerequisites

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

### Procedure

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

```
apiVersion: metal3.io/v1alpha1
kind: BareMetalHost
metadata:
  name: <bare_metal_host_name>
```

```
spec:
  online: true
  bmc:
    address: <bmc_address>
    credentialsName: <secret_credentials_name> ❶
    disableCertificateVerification: True
  bootMACAddress: <host_boot_mac_address>
  hardwareProfile: unknown
```

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

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



#### NOTE

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

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

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

#### Prerequisites

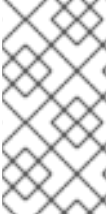
- Install RHCOS bare metal compute machines for use in the cluster, and create corresponding **BareMetalHost** objects.
- Install the OpenShift Container Platform CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

#### Procedure

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

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

Wait for the new scaled machines to start.



## NOTE

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

### Additional resources

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

## CHAPTER 14. WHAT HUGE PAGES DO AND HOW THEY ARE CONSUMED BY APPLICATIONS

### 14.1. 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. In order 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.

### 14.2. 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](#)

## 14.3. 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.

### 14.3.1. 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 15. PERFORMANCE ADDON OPERATOR FOR LOW LATENCY NODES

## 15.1. 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 in order 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.

## 15.2. 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.

### 15.2.1. 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
```

- 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
spec:
  targetNamespaces:
    - 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

```
4.6
```

- 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:

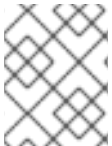
```
$ oc create -f pao-sub.yaml
```

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

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

### 15.2.2. 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, under **A specific namespace on the cluster** select **openshift-performance-addon-operator**. 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-operators** project with a **Status** of **Succeeded**.

**NOTE**

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

If the Operator does not appear as installed, you can 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 **openshift-operators** project.

## 15.3. 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.

### 15.3.1. About upgrading Performance Addon Operator

- You can upgrade to the next minor version of Performance Addon Operator by using the OpenShift 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.

#### 15.3.1.1. 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.

#### 15.3.1.2. 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 4.6, select **4.6**.
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
```

### 15.3.2. 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
4.6.0    performance-addon-operator.v4.5.0          Installing
4.5.0                                         Replacing
```

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

```
# oc get csv
```

#### Example output

```
NAME                                DISPLAY                                VERSION  REPLACES
PHASE
performance-addon-operator.v4.5.0    Performance Addon Operator            4.6.0    performance-
```



addon-operator.v4.5.0 Succeeded

## 15.4. 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.



### WARNING

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

### 15.4.1. Known limitations for real-time

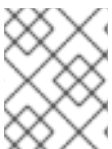


#### NOTE

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.

### 15.4.2. 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. Add the label **worker-rt** to the worker nodes that require the real-time capability by using the **oc** command.
4. Create a new machine config pool for real-time nodes:

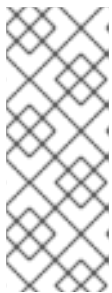
```

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

```

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

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



#### NOTE

You must decide which nodes are configured with real-time workloads. You could configure all of the nodes in the cluster, or a subset of the nodes. The Performance Addon Operator that expects all of the nodes are part of a dedicated machine config pool. If you use all of the nodes, you must point the 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.

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

```

apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: example-performanceprofile
spec:
  ...
  realTimeKernel:
    enabled: true
  nodeSelector:

```

```
node-role.kubernetes.io/worker-rt: ""
machineConfigPoolSelector:
  machineconfiguration.openshift.io/role: worker-rt
```

- Verify that a matching machine config pool exists with a label:

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

### Example output

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

- 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.
- Verify everything is working as expected.

### 15.4.3. 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					
rt-worker-0.example.com	Ready	worker,worker-rt	5d17h	v1.22.1	
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://1.19.0-90.rhaos4.6.git4a0ac05.el8-rc.1			
[...]					

### 15.4.4. Creating a workload that works in real-time

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

#### Procedure

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

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

### 15.4.5. 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:

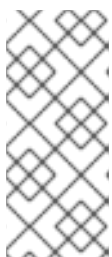
```
$ oc apply -f qos-pod.yaml --namespace=qos-example
```

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.

### 15.4.6. 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/v1
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: "true"
...
...
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.

### 15.4.7. 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](#) .

### 15.4.8. 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](#) .

## 15.5. 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-###: ###
```

## 15.6. 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
```

## 15.7. RESTRICTING CPUS FOR INFRA AND APPLICATION CONTAINERS

Generic housekeeping and workload tasks use CPUs in a way that may impact latency-sensitive processes. By default, the container runtime uses all online CPUs to run all containers together, which can result in context switches and spikes in latency. Partitioning the CPUs prevents noisy processes from interfering with latency-sensitive processes by separating them from each other. The following table describes how processes run on a CPU after you have tuned the node using the Performance Addon Operator:

**Table 15.1. Process' CPU assignments**

Process type	Details
<b>Burstable</b> and <b>BestEffort</b> pods	Runs on any CPU except where low latency workload is running

Process type	Details
Infrastructure pods	Runs on any CPU except where low latency workload is running
Interrupts	Redirects to reserved CPUs (optional in OpenShift Container Platform 4.6 and later)
Kernel processes	Pins to reserved CPUs
Latency-sensitive workload pods	Pins to a specific set of exclusive CPUs from the isolated pool
OS processes/systemd services	Pins to reserved CPUs

The allocatable capacity of cores on a node for pods of all QoS process types, **Burstable**, **BestEffort**, or **Guaranteed**, is equal to the capacity of the isolated pool. The capacity of the reserved pool is removed from the node's total core capacity for use by the cluster and operating system housekeeping duties.

### Example 1

A node features a capacity of 100 cores. Using a performance profile, the cluster administrator allocates 50 cores to the isolated pool and 50 cores to the reserved pool. The cluster administrator assigns 25 cores to QoS **Guaranteed** pods and 25 cores for **BestEffort** or **Burstable** pods. This matches the capacity of the isolated pool.

### Example 2

A node features a capacity of 100 cores. Using a performance profile, the cluster administrator allocates 50 cores to the isolated pool and 50 cores to the reserved pool. The cluster administrator assigns 50 cores to QoS **Guaranteed** pods and one core for **BestEffort** or **Burstable** pods. This exceeds the capacity of the isolated pool by one core. Pod scheduling fails because of insufficient CPU capacity.

The exact partitioning pattern to use depends on many factors like hardware, workload characteristics and the expected system load. Some sample use cases are as follows:

- If the latency-sensitive workload uses specific hardware, such as a network interface card (NIC), ensure that the CPUs in the isolated pool are as close as possible to this hardware. At a minimum, you should place the workload in the same Non-Uniform Memory Access (NUMA) node.
- The reserved pool is used for handling all interrupts. When depending on system networking, allocate a sufficiently-sized reserve pool to handle all the incoming packet interrupts. In 4.6 and later versions, workloads can optionally be labeled as sensitive. The decision regarding which specific CPUs should be used for reserved and isolated partitions requires detailed analysis and measurements. Factors like NUMA affinity of devices and memory play a role. The selection also depends on the workload architecture and the specific use case.



### IMPORTANT

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



To ensure that housekeeping tasks and workloads do not interfere with each other, specify two groups of CPUs in the **spec** section of the performance profile.

- **isolated** - Specifies the CPUs for the application container workloads. Workloads running on these CPUs experience the lowest latency as well as zero interruptions and can, for example, reach high zero packet loss bandwidth.
- **reserved** - Specifies the CPUs for the cluster and operating system housekeeping duties. Threads in the **reserved** group are often busy. Do not run latency-sensitive applications in the **reserved** group. Latency-sensitive applications run in the **isolated** group. .Procedure
  1. Create a performance profile appropriate for the environment's hardware and topology.
  2. Add the **reserved** and **isolated** parameters with the CPUs you want reserved and isolated for the infra and application containers:

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: infra-cpus
spec:
  cpu:
    reserved: "0-4,9" 1
    isolated: "5-8" 2
  nodeSelector: 3
  node-role.kubernetes.io/worker: ""
```

- 1 Specify which CPUs are for infra containers to perform cluster and operating system housekeeping duties.
- 2 Specify which CPUs are for application containers to run workloads.
- 3 Specify a node selector to apply the performance profile to specific nodes.

## 15.8. 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/v1
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](#).

## 15.9. 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
- Having the Performance Addon Operator installed
- Having the SR-IOV Operator installed
- Having the PTP Operator installed
- Using OVN kubernetes as the SDN

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

### 15.9.1. Prerequisites

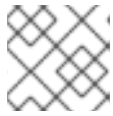
- The test entrypoint is `/usr/bin/test-run.sh`. It runs both a setup test set and the real conformance test suite. The minimum requirement is to provide it with a kubeconfig file and its related `$KUBECONFIG` environment variable, mounted through a volume.
- The tests assumes that a given feature is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

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

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

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

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

**NOTE**

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

**15.9.2. Running the tests**

Assuming the 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.

**15.9.3. 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:v4.6 /usr/bin/test-run.sh
```

**15.9.3.1. 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:v4.6 /usr/bin/test-run.sh -ginkgo.focus="performance|sctp"
```

**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.

**15.9.3.2. Available features**

The set of available features to filter are:

- **performance**
- **sriov**

- **ptp**
- **sctp**
- **dpdk**

#### 15.9.4. 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:v4.6 /usr/bin/test-run.sh -ginkgo.dryRun -ginkgo.v
```

#### 15.9.5. 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.

##### 15.9.5.1. 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:v4.6 /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.

##### 15.9.5.2. 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:v4.6 /usr/bin/test-run.sh
```

##### 15.9.5.3. 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:dppk-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)/:kubecfg -e KUBECONFIG=/kubecfg/kubecfg
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
```

#### 15.9.5.4. 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:4.6"
  },
  {
    "registry": "public.registry.io:5000",
    "image": "imagefordpdk:4.6"
  }
]
```

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 -
```

#### 15.9.6. 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
```

##### 15.9.6.1. 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.

## 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.

### 15.9.6.2. 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
```

### 15.9.6.3. 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/v1
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
```

### 15.9.6.4. 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)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig -e
CLEAN_PERFORMANCE_PROFILE="false" registry.redhat.io/openshift-kni/cnf-tests /usr/bin/test-
run.sh
```

### 15.9.7. Troubleshooting

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

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

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

### 15.9.8. Test reports

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

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

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

#### 15.9.8.3. 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)/:/kubeconfig:Z** to allow podman to do the proper SELinux relabeling.

#### 15.9.8.4. 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.

### 15.9.8.5. 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/v1
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)/:kubecfg:Z -e KUBECONFIG=/kubecfg/kubecfg -e
PERFORMANCE_PROFILE_MANIFEST_OVERRIDE=/kubecfg/manifest.yaml
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.6 /usr/bin/test-run.sh
```

### 15.9.9. 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.

#### 15.9.9.1. 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.

#### 15.9.9.2. 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.

### 15.9.9.3. 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.

### 15.9.9.4. 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.

### 15.9.9.5. 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.

### 15.9.9.6. Cleaning up

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

## 15.10. 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.

### 15.10.1. 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

```

- 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

```

- 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

```

## 15.11. 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.

### 15.11.1. 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.

### 15.11.2. 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 Performance Addon Operator debugging information with **must-gather**, you must specify the Performance Addon Operator **must-gather** image:

```
--image=registry.redhat.io/openshift4/performance-addon-operator-must-gather-rhel8:v4.6.
```

### 15.11.3. 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 the Performance Addon Operator:

```
$ oc adm must-gather \  
--image-stream=openshift/must-gather \ 1  
  
--image=registry.redhat.io/openshift4/performance-addon-operator-must-gather-rhel8:v4.6  
2
```

- 1** The default OpenShift Container Platform **must-gather** image.
  - 2** The **must-gather** image for low latency tuning diagnostics.
3. Create a compressed file from the **must-gather** directory that was created in your working directory. For example, on a computer that uses a Linux operating system, run the following command:

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

- 1** Replace **must-gather-local.5421342344627712289/** with the actual directory name.
4. Attach the compressed file to your support case on the [Red Hat Customer Portal](#).

## Additional resources

- For more information about MachineConfig and KubeletConfig, see [Managing nodes](#).
- For more information about the Node Tuning Operator, see [Using the Node Tuning Operator](#).
- For more information about the PerformanceProfile, see [Configuring huge pages](#).
- For more information about consuming huge pages from your containers, see [How huge pages are consumed by apps](#).



## CHAPTER 16. OPTIMIZING DATA PLANE PERFORMANCE WITH THE INTEL FPGA PAC N3000 AND INTEL VRAN DEDICATED ACCELERATOR ACC100

### 16.1. UNDERSTANDING INTEL HARDWARE ACCELERATOR CARDS FOR OPENSIFT CONTAINER PLATFORM

Hardware accelerator cards from Intel accelerate 4G/LTE and 5G Virtualized Radio Access Networks (vRAN) workloads. This in turn increases the overall compute capacity of a commercial, off-the-shelf platform.

#### Intel FPGA PAC N3000

The Intel FPGA PAC N3000 is a reference FPGA and uses 4G/LTE or 5G forward error correction (FEC) as an example workload that accelerates the 5G or 4G/LTE RAN layer 1 (L1) base station network function. Flash the Intel FPGA PAC N3000 card with 4G/LTE or 5G bitstreams to support vRAN workloads.

The Intel FPGA PAC N3000 is a full-duplex, 100 Gbps in-system, re-programmable acceleration card for multi-workload networking application acceleration.

When the Intel FPGA PAC N3000 is programmed with a 4G/LTE or 5G bitstream, it exposes the Single Root I/O Virtualization (SR-IOV) virtual function (VF) devices used to accelerate the FEC in the vRAN workload. To take advantage of this functionality for a cloud-native deployment, the physical function (PF) of the device must be bound to the **pf-pci-stub** driver to create several VFs. After the VFs are created, the VFs must be bound to a DPDK userspace driver (vfio) to allocate them to specific pods running the vRAN workload.

Intel FPGA PAC N3000 support on OpenShift Container Platform depends on two Operators:

- OpenNESS Operator for Intel FPGA PAC N3000 (Programming)
- OpenNESS Operator for Wireless FEC Accelerators

#### vRAN Dedicated Accelerator ACC100

The vRAN Dedicated Accelerator ACC100, based on Intel's eASIC technology is designed to offload and accelerate the computing-intensive process of forward error correction for 4G/LTE and 5G technology, freeing up processing power. Intel eASIC devices are structured ASICs, an intermediate technology between FPGAs and standard application-specific integrated circuits (ASICs).

Intel vRAN Dedicated Accelerator ACC100 support on OpenShift Container Platform uses one Operator:

- OpenNESS Operator for Wireless FEC Accelerators

### 16.2. INSTALLING THE OPENNESS OPERATOR FOR INTEL FPGA PAC N3000

The OpenNESS Operator for Intel FPGA PAC N3000 orchestrates and manages the resources or devices exposed by the Intel FPGA PAC N3000 card within the OpenShift Container Platform cluster.

For vRAN use cases, the OpenNESS Operator for Intel FPGA PAC N3000 is used with the OpenNESS Operator for Wireless FEC Accelerators.

As a cluster administrator, you can install the OpenNESS Operator for Intel FPGA PAC N3000 by using the OpenShift Container Platform CLI or the web console.

## 16.2.1. Installing the Operator by using the CLI

As a cluster administrator, you can install the Operator by 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 N3000 Operator by completing the following actions:
  - a. Define the **vran-acceleration-operators** namespace by creating a file named **n3000-namespace.yaml** file as shown in the following example:

```
apiVersion: v1
kind: Namespace
metadata:
  name: vran-acceleration-operators
  labels:
    openshift.io/cluster-monitoring: "true"
```

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

```
$ oc create -f n3000-namespace.yaml
```

2. Install the N3000 Operator in the namespace you created in the previous step:
  - a. Create the following **OperatorGroup** CR and save the YAML in the **n3000-operatorgroup.yaml** file:

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: n3000-operators
  namespace: vran-acceleration-operators
spec:
  targetNamespaces:
    - vran-acceleration-operators
```

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

```
$ oc create -f n3000-operatorgroup.yaml
```

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

```
$ oc get packagemanifest n3000 -n openshift-marketplace -o
jsonpath='{.status.defaultChannel}'
```

### Example output

```
stable
```

- d. Create the following **Subscription** CR and save the YAML in the **n3000-sub.yaml** file:

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: n3000-subscription
  namespace: vran-acceleration-operators
spec:
  channel: "<channel>" 1
  name: n3000
  source: certified-operators 2
  sourceNamespace: openshift-marketplace
```

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

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

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

```
$ oc create -f n3000-sub.yaml
```

### Verification

- Verify the Operator is installed:

```
$ oc get csv
```

### Example output

```
NAME          DISPLAY                               VERSION REPLACES  PHASE
n3000.v1.1.0  OpenNESS Operator for Intel® FPGA PAC N3000  1.1.0
Succeeded
```

You have now successfully installed the Operator.

## 16.2.2. Installing the OpenNESS Operator for Intel FPGA PAC N3000 Operator by using the web console

As a cluster administrator, you can install the OpenNESS Operator for Intel FPGA PAC N3000 by using the web console.

**NOTE**

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

**Procedure**

1. Install the OpenNESS Operator for Intel FPGA PAC N3000 by using the OpenShift Container Platform web console:
  - a. In the OpenShift Container Platform web console, click **Operators** → **OperatorHub**.
  - b. Choose **OpenNESS Operator for Intel FPGA PAC N3000** 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 N3000 Operator is installed successfully:
  - a. Switch to the **Operators** → **Installed Operators** page.
  - b. Ensure that **OpenNESS Operator for Intel FPGA PAC N3000** is listed in the **vran-acceleration-operators** 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 console does not indicate that the Operator is installed, perform the following troubleshooting steps:

- 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 **vran-acceleration-operators** project.

## 16.3. PROGRAMMING THE OPENNESS OPERATOR FOR INTEL FPGA PAC N3000

When the Intel FPGA PAC N3000 is programmed with a vRAN 5G bitstream, the hardware exposes the Intel FPGA PAC N3000 with a vRAN 5G bitstream. This bitstream exposes the Single Root I/O Virtualization (SR-IOV) virtual function (VF) devices used to accelerate the FEC in the vRAN workload.

As a cluster administrator, you can install the OpenNESS Operator for Intel FPGA PAC N3000 by using the OpenShift Container Platform CLI or the web console.

### 16.3.1. Programming the N3000 with a vRAN bitstream

As a cluster administrator, you can program the Intel FPGA PAC N3000 with a vRAN 5G bitstream. This bitstream exposes the Single Root I/O Virtualization (SR-IOV) virtual function (VF) devices that are used to accelerate the forward error correction (FEC) in the vRAN workload.

The role of forward error correction (FEC) is to correct transmission errors, where certain bits in a message can be lost or garbled. Messages can be lost or garbled due to noise in the transmission media, interference, or low signal strength. Without FEC, a garbled message would have to be resent, adding to the network load and impacting both throughput and latency.

### Prerequisites

- Intel FPGA PAC N3000 card
- Performance Addon Operator with RT kernel configuration
- Node or nodes installed with the OpenNESS Operator for Intel FPGA PAC N3000
- Log in as a user with **cluster-admin** privileges



### NOTE

All the commands run in the **vrان-acceleration-operators** namespace.

### Procedure

1. Change to the **vrان-acceleration-operators** project:

```
$ oc project vran-acceleration-operators
```

2. Verify that the pods are running:

```
$ oc get pods
```

### Example output

```
NAME                                READY   STATUS    RESTARTS   AGE
fpga-driver-daemonset-8xz4c         1/1     Running   0           15d
fpgainfo-exporter-vhvdq             1/1     Running   1           15d
N3000-controller-manager-b68475c76-gcc6v  2/2     Running   1           15d
N3000-daemonset-5k55l               1/1     Running   1           15d
N3000-discovery-blmj1               1/1     Running   1           15d
N3000-discovery-lblh7               1/1     Running   1           15d
```

The following section provides information on the installed pods:

- **fpga-driver-daemonset** provides and loads the required Open Programmable Accelerator Engine (OPAE) drivers
- **fpgainfo-exporter** provides N3000 telemetry data for Prometheus
- **N3000-controller-manager** applies N3000Node CRs to the cluster and manages all the operand containers
- **N3000-daemonset** is the main worker application. It monitors the changes in each node's CR and acts on the changes. The logic implemented into this Daemon takes care of updating the cards' FPGA user image and NIC firmware. It is also responsible for draining the nodes and taking them out of commission when required by the update.

- **N3000-discovery** discovers N3000 Accelerator devices installed and labels worker nodes if devices are present

3. Get all the nodes containing the Intel FPGA PAC N3000 card:

```
$ oc get n3000node
```

### Example output

```
NAME          FLASH
node1         NotRequested
```

4. Get information about the card on each node:

```
$ oc get n3000node node1 -o yaml
```

### Example output

```
status:
  conditions:
  - lastTransitionTime: "2020-12-15T17:09:26Z"
    message: Inventory up to date
    observedGeneration: 1
    reason: NotRequested
    status: "False"
    type: Flashed
  fortville:
  - N3000PCI: 0000:1b:00.0
    NICs:
    - MAC: 64:4c:36:11:1b:a8
      NVMVersion: 7.00 0x800052b0 0.0.0
      PCIAddr: 0000:1a:00.0
      name: Ethernet Controller XXV710 Intel(R) FPGA Programmable Acceleration Card
N3000 for Networking
    - MAC: 64:4c:36:11:1b:a9
      NVMVersion: 7.00 0x800052b0 0.0.0
      PCIAddr: 0000:1a:00.1
      name: Ethernet Controller XXV710 Intel(R) FPGA Programmable Acceleration Card
N3000 for Networking
    - MAC: 64:4c:36:11:1b:ac
      NVMVersion: 7.00 0x800052b0 0.0.0
      PCIAddr: 0000:1c:00.0
      name: Ethernet Controller XXV710 Intel(R) FPGA Programmable Acceleration Card
N3000 for Networking
    - MAC: 64:4c:36:11:1b:ad
      NVMVersion: 7.00 0x800052b0 0.0.0
      PCIAddr: 0000:1c:00.1
      name: Ethernet Controller XXV710 Intel(R) FPGA Programmable Acceleration Card
N3000 for Networking
  fpga:
  - PCIAddr: 0000:1b:00.0 1
    bitstreamId: "0x23000410010310" 2
    bitstreamVersion: 0.2.3
    deviceId: "0x0b30"
```

- 1 The **PCIAddr** field indicates the PCI address of the card.
- 2 The **bitstreamId** field indicates the bitstream that is currently stored in flash.

5. Save the current **bitstreamId**, **PCIAddr**, the name, and the **deviceId** without "0x" padding.

```
$ oc get n3000node -o json
```

6. Update the user bitstream of the Intel FPGA PAC N3000 card:

- a. Define the N3000 cluster resource to program by creating a file named **n3000-cluster.yaml** as shown in the following example:

```
apiVersion: fpga.intel.com/v1
kind: N3000Cluster
metadata:
  name: n3000 1
  namespace: vran-acceleration-operators
spec:
  nodes:
    - nodeName: "node1" 2
      fpga:
        - userImageURL: "http://10.10.10.122:8000/pkg/20ww27.5-2x2x25G-5GLDPC-
v1.6.1-3.0.0_unsigned.bin" 3
          PCIAddr: "0000:1b:00.0" 4
          checksum: "0b0a87b974d35ea16023ceb57f7d5d9c" 5
```

- 1 Specify the name. The name must be **n3000**.
- 2 Specify the node to program.
- 3 Specify the URL for the user bitstream. This bitstream file must be accessible on an HTTP or HTTPS server.
- 4 Specify the PCI address of the card to program.
- 5 Specify the MD5 checksum of the bitstream that is specified in the **userImageURL** field.

The N3000 daemon updates the FPGA user bitstream using the Open Programmable Acceleration Engine (OPAE) tools and resets the PCI device. The update of the FPGA user bitstream can require up to 40 minutes per card. For programming cards on multiple nodes, the programming happens one node at a time.

- b. Apply the update to begin programming the card with the bitstream:

```
$ oc apply -f n3000-cluster.yaml
```

The N3000 daemon starts programming the bitstream after the appropriate 5G FEC user bitstream has been provisioned, such as **20ww27.5-2x2x25G-5GLDPC-v1.6.1-3.0.0\_unsigned.bin** in this example, and after the CR has been created.

- c. Check the status:

■

```
oc get n3000node
```

### Example output

```
NAME          FLASH
node1        InProgress
```

7. Check the logs:

a. Determine the pod name of the N3000 daemon:

```
$ oc get pod -o wide | grep n3000-daemonset | grep node1
```

### Example output

```
n3000-daemonset-5k55l          1/1   Running   0          15d
```

b. View the logs:

```
$ oc logs n3000-daemonset-5k55l
```

### Example output

```
...
{"level":"info","ts":1608054338.8866854,"logger":"daemon.drainhelper.cordonAndDrain()","msg":"node drained"}
{"level":"info","ts":1608054338.8867319,"logger":"daemon.drainhelper.Run()","msg":"worker function - start"}
{"level":"info","ts":1608054338.9003832,"logger":"daemon.fpgaManager.ProgramFPGAs","msg":"Start program","PCIAddr":"0000:1b:00.0"}
{"level":"info","ts":1608054338.9004142,"logger":"daemon.fpgaManager.ProgramFPGA","msg":"Starting","pci":"0000:1b:00.0"}
{"level":"info","ts":1608056309.9367146,"logger":"daemon.fpgaManager.ProgramFPGA","msg":"Program FPGA completed, start new power cycle N3000 ...","pci":"0000:1b:00.0"}
{"level":"info","ts":1608056333.3528838,"logger":"daemon.drainhelper.Run()","msg":"worker function - end","performUncordon":true}
...
```

The log file indicates the following flow of events:

- The bitstream is downloaded and validated.
- The node is drained and no workload is able to run during this time.
- Flashing is started:
  - The bitstream is flashed into the card.
  - The bitstream is applied.
- After flashing is complete the PCI device or devices on the node or nodes are reloaded. The OpenNESS SR-IOV Operator for Wireless FEC Accelerators is now able to find the new flashed device or devices.



## Verification

1. Verify the status after the FPGA user bitstream update is complete:

```
oc get n3000node
```

### Example output

```
NAME      FLASH
node1     Succeeded
```

2. Verify that the bitstream ID of the card has changed:

```
oc get n3000node node1 -o yaml
```

### Example output

```
status:
  conditions:
    - lastTransitionTime: "2020-12-15T18:18:53Z"
      message: Flashed successfully 1
      observedGeneration: 2
      reason: Succeeded
      status: "True"
      type: Flashed
  fortville:
    - N3000PCI: 0000:1b:00.0
      NICs:
        - MAC: 64:4c:36:11:1b:a8
          NVMVersion: 7.00 0x800052b0 0.0.0
          PCIAddr: 0000:1a:00.0
          name: Ethernet Controller XXV710 Intel(R) FPGA Programmable Acceleration Card
            N3000 for Networking
          - MAC: 64:4c:36:11:1b:a9
            NVMVersion: 7.00 0x800052b0 0.0.0
            PCIAddr: 0000:1a:00.1
            name: Ethernet Controller XXV710 Intel(R) FPGA Programmable Acceleration Card
              N3000 for Networking
            - MAC: 64:4c:36:11:1b:ac
              NVMVersion: 7.00 0x800052b0 0.0.0
              PCIAddr: 0000:1c:00.0
              name: Ethernet Controller XXV710 Intel(R) FPGA Programmable Acceleration Card
                N3000 for Networking
            - MAC: 64:4c:36:11:1b:ad
              NVMVersion: 7.00 0x800052b0 0.0.0
              PCIAddr: 0000:1c:00.1
              name: Ethernet Controller XXV710 Intel(R) FPGA Programmable Acceleration Card
                N3000 for Networking
          fpga:
            - PCIAddr: 0000:1b:00.0 2
              bitstreamId: "0x2315842A010601" 3
              bitstreamVersion: 0.2.3
              deviceId: "0x0b30" 4
```

- 1 The **message** field indicates the device is successfully flashed.
- 2 The **PCIAddr** field indicates the PCI address of the card.
- 3 The **bitstreamId** field indicates the updated bitstream ID.
- 4 The **deviceID** field indicates that device ID of the bitstream inside the card exposed to the system.

3. Check the FEC PCI devices on the node:

- a. Verify the node configuration is applied correctly:

```
$ oc debug node/node1
```

**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#
```

- b. Verify that you can use the node file system:

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

**Expected output**

```
sh-4.4#
```

- c. List the PCI devices associated with the accelerator on your system:

```
$ lspci | grep accelerators
```

**Expected output**

```
1b:00.0 Processing accelerators: Intel Corporation Device 0b30
1d:00.0 Processing accelerators: Intel Corporation Device 0d8f (rev 01)
```

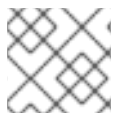
Devices belonging to the FPGA are reported in the output. Device ID **0b30** is the RSU interface used to program the card, and the **0d8f** is a physical function of the newly programmed 5G device.

## 16.4. INSTALLING THE OPENNESS SR-IOV OPERATOR FOR WIRELESS FEC ACCELERATORS

The role of the OpenNESS SR-IOV Operator for Wireless FEC Accelerators is to orchestrate and manage the devices exposed by a range of Intel vRAN FEC acceleration hardware within the OpenShift Container Platform cluster.

One of the most compute-intensive 4G/LTE and 5G workloads is RAN layer 1 (L1) forward error correction (FEC). FEC resolves data transmission errors over unreliable or noisy communication channels. FEC technology detects and corrects a limited number of errors in 4G/LTE or 5G data without the need for retransmission.

The FEC devices are provided by the Intel FPGA PAC N3000 and the Intel vRAN Dedicated Accelerator ACC100 for the vRAN use case.



**NOTE**

The Intel FPGA PAC N3000 FPGA requires flashing with an 4G/LTE or 5G bitstream.

The OpenNESS SR-IOV Operator for Wireless FEC Accelerators provides functionality to create virtual functions (VFs) for the FEC device, binds them to appropriate drivers, and configures the VFs queues for functionality in 4G/LTE or 5G deployment.

As a cluster administrator, you can install the OpenNESS SR-IOV Operator for Wireless FEC Accelerators by using the OpenShift Container Platform CLI or the web console.

### 16.4.1. Installing the OpenNESS SR-IOV Operator for Wireless FEC Accelerators by using the CLI

As a cluster administrator, you can install the OpenNESS SR-IOV Operator for Wireless FEC Accelerators by 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 OpenNESS SR-IOV Operator for Wireless FEC Accelerators by completing the following actions:
  - a. Define the **vran-acceleration-operators** namespace by creating a file named **sriov-namespace.yaml** as shown in the following example:

```
apiVersion: v1
kind: Namespace
metadata:
  name: vran-acceleration-operators
  labels:
    openshift.io/cluster-monitoring: "true"
```

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

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

2. Install the OpenNESS SR-IOV Operator for Wireless FEC Accelerators 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 **sriov-operatorgroup.yaml** file:

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: vran-operators
  namespace: vran-acceleration-operators
spec:
  targetNamespaces:
    - vran-acceleration-operators
```

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

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

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

```
$ oc get packagemanifest sriov-fec -n openshift-marketplace -o
jsonpath='{.status.defaultChannel}'
```

### Example output

```
stable
```

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

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: sriov-fec-subscription
  namespace: vran-acceleration-operators
spec:
  channel: "<channel>" 1
  name: sriov-fec
  source: certified-operators 2
  sourceNamespace: openshift-marketplace
```

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

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

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

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

### Verification

- Verify that the Operator is installed:

```
$ oc get csv -n vran-acceleration-operators -o custom-
columns=Name:.metadata.name,Phase:.status.phase
```

### Example output

Name	Phase
sriov-fec.v1.1.0	Succeeded

## 16.4.2. Installing the OpenNESS SR-IOV Operator for Wireless FEC Accelerators by using the web console

As a cluster administrator, you can install the OpenNESS SR-IOV Operator for Wireless FEC Accelerators by using the web console.



### NOTE

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

### Procedure

1. Install the OpenNESS SR-IOV Operator for Wireless FEC Accelerators by using the OpenShift Container Platform web console:
  - a. In the OpenShift Container Platform web console, click **Operators** → **OperatorHub**.
  - b. Choose **OpenNESS SR-IOV Operator for Wireless FEC Accelerators** 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 SRIOV-FEC Operator is installed successfully:
  - a. Switch to the **Operators** → **Installed Operators** page.
  - b. Ensure that **OpenNESS SR-IOV Operator for Wireless FEC Accelerators** is listed in the **vran-acceleration-operators** 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 console does not indicate that the Operator is installed, perform the following troubleshooting steps:

- 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 **vran-acceleration-operators** project.

### 16.4.3. Configuring the SR-IOV-FEC Operator for Intel FPGA PAC N3000

This section describes how to program the SR-IOV-FEC Operator for Intel FPGA PAC N3000. The SR-IOV-FEC Operator handles the management of the forward error correction (FEC) devices that are used to accelerate the FEC process in vRAN L1 applications.

Configuring the SR-IOV-FEC Operator involves:

- Creating the desired virtual functions (VFs) for the FEC device
- Binding the VFs to the appropriate drivers
- Configuring the VF queues for desired functionality in a 4G or 5G deployment

The role of forward error correction (FEC) is to correct transmission errors, where certain bits in a message can be lost or garbled. Messages can be lost or garbled due to noise in the transmission media, interference, or low signal strength. Without FEC, a garbled message would have to be resent, adding to the network load and impacting throughput and latency.

#### Prerequisites

- Intel FPGA PAC N3000 card
- Node or nodes installed with the OpenNESS Operator for Intel FPGA PAC N3000 (Programming)
- Node or nodes installed with the OpenNESS Operator for Wireless FEC Accelerators
- RT kernel configured with Performance Addon Operator

#### Procedure

1. Change to the **vran-acceleration-operators** project:

```
$ oc project vran-acceleration-operators
```

2. Verify that the SR-IOV-FEC Operator is installed:

```
$ oc get csv -o custom-columns=Name:.metadata.name,Phase:.status.phase
```

#### Example output

Name	Phase
sriov-fec.v1.1.0	Succeeded
n3000.v1.1.0	Succeeded

3. Verify that the **N3000** and **sriov-fec** pods are running:

```
$ oc get pods
```

#### Example output

NAME	READY	STATUS	RESTARTS	AGE
fpga-driver-daemonset-8xz4c	1/1	Running	0	15d

fpgainfo-exporter-vhvdq	1/1	Running	1	15d	
N3000-controller-manager-b68475c76-gcc6v	2/2	Running	1	15d	15d
N3000-daemonset-5k55l	1/1	Running	1	15d	
N3000-discovery-blmjl	1/1	Running	1	15d	
N3000-discovery-lblh7	1/1	Running	1	15d	
sriov-device-plugin-j5jlv	1/1	Running	1	15d	
sriov-fec-controller-manager-85b6b8f4d4-gd2qg	1/1	Running	1	15d	15d
sriov-fec-daemonset-kqqs6	1/1	Running	1	15d	

The following section provides information on the installed pods:

- **fpga-driver-daemonset** provides and loads the required Open Programmable Accelerator Engine (OPAE) drivers
- **fpgainfo-exporter** provides N3000 telemetry data for Prometheus
- **N3000-controller-manager** applies N3000Node CRs to the cluster and manages all the operand containers
- **N3000-daemonset** is the main worker application
- **N3000-discovery** discovers N3000 Accelerator devices installed and labels worker nodes if devices are present
- **sriov-device-plugin** expose the FEC virtual functions as resources under the node
- **sriov-fec-controller-manager** applies CR to the node and maintains the operands containers
- **sriov-fec-daemonset** is responsible for:
  - Discovering the SRIOV NICs on each node.
  - Syncing the status of the custom resource (CR) defined in step 6.
  - Taking the spec of the CR as input and configuring the discovered NICs.

4. Retrieve all the nodes containing one of the supported vRAN FEC accelerator devices:

```
$ oc get sriovfecnodeconfig
```

### Example output

```
NAME          CONFIGURED
node1         Succeeded
```

5. Find the physical function (PF) of the SR-IOV FEC accelerator device to configure:

```
$ oc get sriovfecnodeconfig node1 -o yaml
```

### Example output

```
status:
  conditions:
  - lastTransitionTime: "2021-03-19T17:19:37Z"
```

```

message: Configured successfully
observedGeneration: 1
reason: ConfigurationSucceeded
status: "True"
type: Configured
inventory:
  sriovAccelerators:
  - deviceID: 0d5c
    driver: ""
    maxVirtualFunctions: 16
    pciAddress: 0000.1d.00.0 1
    vendorID: "8086"
    virtualFunctions: [] 2

```

- 1** This field indicates the PCI Address of the card.
- 2** This field shows that the virtual functions are empty.

6. Configure the FEC device with the desired setting.

- a. Create the following custom resource (CR) and save the YAML in the **sriovfec\_n3000\_cr.yaml** file:

```

apiVersion: sriovfec.intel.com/v1
kind: SriovFecClusterConfig
metadata:
  name: config
  namespace: vran-acceleration-operators
spec:
  nodes:
  - nodeName: node1 1
    physicalFunctions:
    - pciAddress: 0000:1d:00.0 2
      pfDriver: pci-pf-stub
      vfDriver: vfio-pci
      vfAmount: 2 3
      bbDevConfig:
        n3000:
          # Network Type: either "FPGA_5GNR" or "FPGA_LTE"
          networkType: "FPGA_5GNR"
          pfMode: false
          flrTimeout: 610
          downlink:
            bandwidth: 3
            loadBalance: 128
          queues: 4
            vf0: 16
            vf1: 16
            vf2: 0
            vf3: 0
            vf4: 0
            vf5: 0
            vf6: 0
            vf7: 0

```



```
uplink:
bandwidth: 3
loadBalance: 128
queues: 5
vf0: 16
vf1: 16
vf2: 0
vf3: 0
vf4: 0
vf5: 0
vf6: 0
vf7: 0
```

- 1 Specify the node name.
- 2 Specify the PCI Address of the card on which the SR-IOV-FEC Operator will be installed.
- 3 Specify the number of virtual functions. Create two virtual functions.
- 4 On **vf0** create one queue with 16 buses (downlink and uplink).
- 5 On **vf1** create one queue with 16 buses (downlink and uplink).



**NOTE**

For Intel PAC N3000 for vRAN Acceleration the user can create up to 8 VF devices. Each FEC PF device provides a total of 64 queues to be configured, 32 queues for uplink and 32 queues for downlink. The queues would be typically distributed evenly across the VFs.

b. Apply the CR:

```
$ oc apply -f sriovfec_n3000_cr.yaml
```

After applying the CR, the SR-IOV FEC daemon starts configuring the FEC device.

**Verification**

1. Check the status:

```
$ oc get sriovfecclusterconfig config -o yaml
```

**Example output**

```
status:
conditions:
- lastTransitionTime: "2020-12-15T17:19:37Z"
message: Configured successfully
observedGeneration: 1
reason: ConfigurationSucceeded
status: "True"
type: Configured
```

```

inventory:
  sriovAccelerators:
  - deviceID: 0d8f
    driver: pci-pf-stub
    maxVirtualFunctions: 8
    pciAddress: 0000:1d:00.0
    vendorID: "8086"
  virtualFunctions:
  - deviceID: 0d90
    driver: vfio-pci
    pciAddress: 0000:1d:00.1
  - deviceID: 0d90
    driver: vfio-pci
    pciAddress: 0000:1d:00.2

```

## 2. Check the logs:

- a. Determine the name of the SR-IOV daemon pod:

```
$ oc get pod | grep sriov-fec-daemonset
```

### Example output

```
sriov-fec-daemonset-kqqs6          1/1   Running   0      19h
```

- b. View the logs:

```
$ oc logs sriov-fec-daemonset-kqqs6
```

### Example output

```

2020-12-16T12:46:47.720Z      INFO    daemon.NodeConfigurator.applyConfig
configuring PF {"requestedConfig": {"pciAddress":"0000:1d:00.0","pfDriver":"pci-pf-
stub","vfDriver":"vfio-pci","vfAmount":2,"bbDevConfig":{"n3000":{"
"networkType":"FPGA_5GMR","pfMode":false,"flrTimeout":610,"downlink":
{"bandwidth":3,"loadBalance":128,"queues":{"vf0":16,"vf1":16}},"uplink":
{"bandwidth":3,"loadBalance":128,"queues":{"vf0":16,"vf1":16}}}}}
2020-12-16T12:46:47.720Z      INFO    daemon.NodeConfigurator.loadModule
executing command {"cmd": "/usr/sbin/chroot /host/ modprobe pci-pf-stub"}
2020-12-16T12:46:47.724Z      INFO    daemon.NodeConfigurator.loadModule
commands output {"output": ""}
2020-12-16T12:46:47.724Z      INFO    daemon.NodeConfigurator.loadModule
executing command {"cmd": "/usr/sbin/chroot /host/ modprobe vfio-pci"}
2020-12-16T12:46:47.727Z      INFO    daemon.NodeConfigurator.loadModule
commands output {"output": ""}
2020-12-16T12:46:47.727Z      INFO    daemon.NodeConfigurator device's
driver_override path {"path": "/sys/bus/pci/devices/0000:1d:00.0/driver_override"}
2020-12-16T12:46:47.727Z      INFO    daemon.NodeConfigurator driver bind path
{"path": "/sys/bus/pci/drivers/pci-pf-stub/bind"}
2020-12-16T12:46:47.998Z      INFO    daemon.NodeConfigurator device's
driver_override path {"path": "/sys/bus/pci/devices/0000:1d:00.1/driver_override"}
2020-12-16T12:46:47.998Z      INFO    daemon.NodeConfigurator driver bind path
{"path": "/sys/bus/pci/drivers/vfio-pci/bind"}
2020-12-16T12:46:47.998Z      INFO    daemon.NodeConfigurator device's

```

```

driver_override path {"path": "/sys/bus/pci/devices/0000:1d:00.2/driver_override"}
2020-12-16T12:46:47.998Z INFO daemon.NodeConfigurator driver bind path
{"path": "/sys/bus/pci/drivers/vfio-pci/bind"}
2020-12-16T12:46:47.999Z INFO daemon.NodeConfigurator.applyConfig
executing command {"cmd": "/sriov_workdir/pf_bb_config FPGA_5G NR -c
/sriov_artifacts/0000:1d:00.0.ini -p 0000:1d:00.0"}
2020-12-16T12:46:48.017Z INFO daemon.NodeConfigurator.applyConfig
commands output {"output": "ERROR: Section (FLR) or name (flr_time_out) is not valid.
FEC FPGA RTL v3.0
UL.DL Weights = 3.3
UL.DL Load Balance = 1
28.128
Queue-PF/VF Mapping Table = READY
Ring Descriptor Size = 256 bytes

```

	PF	VF0	VF1	VF2	VF3	VF4	VF5	VF6	VF7
UL-Q'00	X								
UL-Q'01	X								
UL-Q'02	X								
UL-Q'03	X								
UL-Q'04	X								
UL-Q'05	X								
UL-Q'06	X								
UL-Q'07	X								
UL-Q'08	X								
UL-Q'09	X								
UL-Q'10	X								
UL-Q'11	X								
UL-Q'12	X								
UL-Q'13	X								
UL-Q'14	X								
UL-Q'15	X								
UL-Q'16		X							
UL-Q'17		X							
UL-Q'18		X							
UL-Q'19		X							
UL-Q'20		X							
UL-Q'21		X							
UL-Q'22		X							
UL-Q'23		X							
UL-Q'24		X							
UL-Q'25		X							
UL-Q'26		X							
UL-Q'27		X							
UL-Q'28		X							
UL-Q'29		X							
UL-Q'30		X							
UL-Q'31		X							
DL-Q'32	X								
DL-Q'33	X								
DL-Q'34	X								
DL-Q'35	X								
DL-Q'36	X								
DL-Q'37	X								

```

DL-Q'38 | | X | | | | | | | |
DL-Q'39 | | X | | | | | | | |
DL-Q'40 | | X | | | | | | | |
DL-Q'41 | | X | | | | | | | |
DL-Q'42 | | X | | | | | | | |
DL-Q'43 | | X | | | | | | | |
DL-Q'44 | | X | | | | | | | |
DL-Q'45 | | X | | | | | | | |
DL-Q'46 | | X | | | | | | | |
DL-Q'47 | | X | | | | | | | |
DL-Q'48 | | | X | | | | | | | |
DL-Q'49 | | | X | | | | | | | |
DL-Q'50 | | | X | | | | | | | |
DL-Q'51 | | | X | | | | | | | |
DL-Q'52 | | | X | | | | | | | |
DL-Q'53 | | | X | | | | | | | |
DL-Q'54 | | | X | | | | | | | |
DL-Q'55 | | | X | | | | | | | |
DL-Q'56 | | | X | | | | | | | |
DL-Q'57 | | | X | | | | | | | |
DL-Q'58 | | | X | | | | | | | |
DL-Q'59 | | | X | | | | | | | |
DL-Q'60 | | | X | | | | | | | |
DL-Q'61 | | | X | | | | | | | |
DL-Q'62 | | | X | | | | | | | |
DL-Q'63 | | | X | | | | | | | |

```

```

-----+-----+-----+-----+-----+-----+-----+-----+-----+
Mode of operation = VF-mode
FPGA_5G NR PF [0000:1d:00.0] configuration complete!"
2020-12-16T12:46:48.017Z INFO daemon.NodeConfigurator.enableMasterBus
executing command {"cmd": "/usr/sbin/chroot /host/ setpci -v -s 0000:1d:00.0
COMMAND"}
2020-12-16T12:46:48.037Z INFO daemon.NodeConfigurator.enableMasterBus
commands output {"output": "0000:1d:00.0 @04 = 0102\n"}
2020-12-16T12:46:48.037Z INFO daemon.NodeConfigurator.enableMasterBus
executing command {"cmd": "/usr/sbin/chroot /host/ setpci -v -s 0000:1d:00.0
COMMAND=0106"}
2020-12-16T12:46:48.054Z INFO daemon.NodeConfigurator.enableMasterBus
commands output {"output": "0000:1d:00.0 @04 0106\n"}
2020-12-16T12:46:48.054Z INFO daemon.NodeConfigurator.enableMasterBus
MasterBus set {"pci": "0000:1d:00.0", "output": "0000:1d:00.0 @04 0106\n"}
2020-12-16T12:46:48.160Z INFO daemon.drainhelper.Run() worker function -
end {"performUncordon": true}

```

3. Check the FEC configuration of the card:

```
$ oc get sriovfecnodeconfig node1 -o yaml
```

### Example output

```

status:
  conditions:
  - lastTransitionTime: "2020-12-15T17:19:37Z"
    message: Configured successfully

```

```

observedGeneration: 1
reason: ConfigurationSucceeded
status: "True"
type: Configured
inventory:
sriovAccelerators:
- deviceID: 0d8f 1
  driver: pci-pf-stub
  maxVirtualFunctions: 8
  pciAddress: 0000:1d:00.0
  vendorID: "8086"
virtualFunctions:
- deviceID: 0d90 2
  driver: vfio-pci
  pciAddress: 0000:1d:00.1
- deviceID: 0d90
  driver: vfio-pci
  pciAddress: 0000:1d:00.2

```

1 0d8f is the **deviceID** physical function of the FEC device.

2 0d90 is the **deviceID** virtual function of the FEC device.

#### 16.4.4. Configuring the SR-IOV-FEC Operator for the Intel vRAN Dedicated Accelerator ACC100

Programming the Intel vRAN Dedicated Accelerator ACC100 exposes the Single Root I/O Virtualization (SRIOV) virtual function (VF) devices that are then used to accelerate the FEC in the vRAN workload. The Intel vRAN Dedicated Accelerator ACC100 accelerates 4G and 5G Virtualized Radio Access Networks (vRAN) workloads. This in turn increases the overall compute capacity of a commercial, off-the-shelf platform. This device is also known as Mount Bryce.

The SR-IOV-FEC Operator handles the management of the forward error correction (FEC) devices that are used to accelerate the FEC process in vRAN L1 applications.

Configuring the SR-IOV-FEC Operator involves:

- Creating the virtual functions (VFs) for the FEC device
- Binding the VFs to the appropriate drivers
- Configuring the VF queues for desired functionality in a 4G or 5G deployment

The role of forward error correction (FEC) is to correct transmission errors, where certain bits in a message can be lost or garbled. Messages can be lost or garbled due to noise in the transmission media, interference, or low signal strength. Without FEC, a garbled message would have to be resent, adding to the network load and impacting throughput and latency.

##### Prerequisites

- Intel FPGA ACC100 5G/4G card
- Node or nodes installed with the OpenNESS Operator for Wireless FEC Accelerators
- Enable global SR-IOV and VT-d settings in the BIOS for the node

- RT kernel configured with Performance Addon Operator
- Log in as a user with **cluster-admin** privileges

## Procedure

1. Change to the **vran-acceleration-operators** project:

```
$ oc project vran-acceleration-operators
```

2. Verify that the SR-IOV-FEC Operator is installed:

```
$ oc get csv -o custom-columns=Name:.metadata.name,Phase:.status.phase
```

### Example output

Name	Phase
sriov-fec.v1.1.0	Succeeded

3. Verify that the **sriov-fec** pods are running:

```
$ oc get pods
```

### Example output

NAME	READY	STATUS	RESTARTS	AGE
sriov-device-plugin-j5jlv	1/1	Running	1	15d
sriov-fec-controller-manager-85b6b8f4d4-gd2qg	1/1	Running	1	15d
sriov-fec-daemonset-kqqs6	1/1	Running	1	15d

- **sriov-device-plugin** expose the FEC virtual functions as resources under the node
  - **sriov-fec-controller-manager** applies CR to the node and maintains the operands containers
  - **sriov-fec-daemonset** is responsible for:
    - Discovering the SRIOV NICs on each node.
    - Syncing the status of the custom resource (CR) defined in step 6.
    - Taking the spec of the CR as input and configuring the discovered NICs.
4. Retrieve all the nodes containing one of the supported vRAN FEC accelerator devices:

```
$ oc get sriovfecnodeconfig
```

### Example output

NAME	CONFIGURED
node1	Succeeded

5. Find the physical function (PF) of the SR-IOV FEC accelerator device to configure:

```
$ oc get sriovfecnodeconfig node1 -o yaml
```

### Example output

```
status:
  conditions:
  - lastTransitionTime: "2021-03-19T17:19:37Z"
    message: Configured successfully
    observedGeneration: 1
    reason: ConfigurationSucceeded
    status: "True"
    type: Configured
  inventory:
    sriovAccelerators:
    - deviceID: 0d5c
      driver: ""
      maxVirtualFunctions: 16
      pciAddress: 0000:af:00.0 ❶
      vendorID: "8086"
      virtualFunctions: [] ❷
```

- ❶ This field indicates the PCI address of the card.
- ❷ This field shows that the virtual functions are empty.

6. Configure the number of virtual functions and queue groups on the FEC device:
  - a. Create the following custom resource (CR) and save the YAML in the **sriovfec\_acc100cr.yaml** file:



#### NOTE

This example configures the ACC100 8/8 queue groups for 5G, 4 queue groups for Uplink, and another 4 queue groups for Downlink.

```
apiVersion: sriovfec.intel.com/v1
kind: SrioVfecClusterConfig
metadata:
  name: config ❶
spec:
  nodes:
  - nodeName: node1 ❷
    physicalFunctions:
    - pciAddress: 0000:af:00.0 ❸
      pfDriver: "pci-pf-stub"
      vfDriver: "vfio-pci"
      vfAmount: 16 ❹
      bbDevConfig:
        acc100:
          # Programming mode: 0 = VF Programming, 1 = PF Programming
          pfMode: false
          numVfBundles: 16
```

```

maxQueueSize: 1024
uplink4G:
  numQueueGroups: 0
  numAqsPerGroups: 16
  aqDepthLog2: 4
downlink4G:
  numQueueGroups: 0
  numAqsPerGroups: 16
  aqDepthLog2: 4
uplink5G:
  numQueueGroups: 4
  numAqsPerGroups: 16
  aqDepthLog2: 4
downlink5G:
  numQueueGroups: 4
  numAqsPerGroups: 16
  aqDepthLog2: 4

```

- 1 Specify a name for the CR object. The only name that can be specified is **config**.
- 2 Specify the node name.
- 3 Specify the PCI address of the card on which the SR-IOV-FEC Operator will be installed.
- 4 Specify the number of virtual functions to create. For the Intel vRAN Dedicated Accelerator ACC100, create all 16 VFs.



#### NOTE

The card is configured to provide up to 8 queue groups with up to 16 queues per group. The queue groups can be divided between groups allocated to 5G and 4G and Uplink and Downlink. The Intel vRAN Dedicated Accelerator ACC100 can be configured for:

- 4G or 5G only
- 4G and 5G at the same time

Each configured VF has access to all the queues. Each of the queue groups have a distinct priority level. The request for a given queue group is made from the application level that is, the vRAN application leveraging the FEC device.

- b. Apply the CR:

```
$ oc apply -f sriovfec_acc100cr.yaml
```

After applying the CR, the SR-IOV FEC daemon starts configuring the FEC device.

#### Verification

1. Check the status:



```
$ oc get sriovfecclusterconfig config -o yaml
```

### Example output

```
status:
  conditions:
  - lastTransitionTime: "2021-03-19T11:46:22Z"
    message: Configured successfully
    observedGeneration: 1
    reason: Succeeded
    status: "True"
    type: Configured
  inventory:
    sriovAccelerators:
    - deviceId: 0d5c
      driver: pci-pf-stub
      maxVirtualFunctions: 16
      pciAddress: 0000:af:00.0
      vendorID: "8086"
      virtualFunctions:
      - deviceId: 0d5d
        driver: vfio-pci
        pciAddress: 0000:b0:00.0
      - deviceId: 0d5d
        driver: vfio-pci
        pciAddress: 0000:b0:00.1
      - deviceId: 0d5d
        driver: vfio-pci
        pciAddress: 0000:b0:00.2
      - deviceId: 0d5d
        driver: vfio-pci
        pciAddress: 0000:b0:00.3
      - deviceId: 0d5d
        driver: vfio-pci
        pciAddress: 0000:b0:00.4
```

2. Check the logs:
  - a. Determine the pod name of the SR-IOV daemon:

```
$ oc get po -o wide | grep sriov-fec-daemonset | grep node1
```

### Example output

```
sriov-fec-daemonset-kqqs6          1/1   Running   0          19h
```

- b. View the logs:

```
$ oc logs sriov-fec-daemonset-kqqs6
```

### Example output

```
{"level": "Level(-
```

```

2)", "ts":1616794345.4786215, "logger": "daemon.drainhelper.cordonAndDrain()", "msg": "no
de drained"}
{"level": "Level(-
4)", "ts":1616794345.4786265, "logger": "daemon.drainhelper.Run()", "msg": "worker
function - start"}
{"level": "Level(-
4)", "ts":1616794345.5762916, "logger": "daemon.NodeConfigurator.applyConfig", "msg": "cur
rent node status", "inventory": {"sriovAccelerat
ors":
[{"vendorID": "8086", "deviceID": "0b32", "pciAddress": "0000:20:00.0", "driver": "", "maxVirtualF
unctions": 1, "virtualFunctions": []}, {"vendorID": "8086"
, "deviceID": "0d5c", "pciAddress": "0000:af:00.0", "driver": "", "maxVirtualFunctions": 16, "virtualF
unctions": []}]}
{"level": "Level(-
4)", "ts":1616794345.5763638, "logger": "daemon.NodeConfigurator.applyConfig", "msg": "co
nfiguring PF", "requestedConfig": {"pciAddress": "
0000:af:00.0", "pfDriver": "pci-pf-stub", "vfDriver": "vfio-pci", "vfAmount": 2, "bbDevConfig":
{"acc100": {"pfMode": false, "numVfBundles": 16, "maxQueueSize": 1
024, "uplink4G":
{"numQueueGroups": 4, "numAqsPerGroups": 16, "aqDepthLog2": 4}, "downlink4G":
{"numQueueGroups": 4, "numAqsPerGroups": 16, "aqDepthLog2": 4}, "uplink5G":
{"numQueueGroups": 0, "numAqsPerGroups": 16, "aqDepthLog2": 4}, "downlink5G":
{"numQueueGroups": 0, "numAqsPerGroups": 16, "aqDepthLog2": 4}}}}
{"level": "Level(-
4)", "ts":1616794345.5774765, "logger": "daemon.NodeConfigurator.loadModule", "msg": "ex
ecuting command", "cmd": "/usr/sbin/chroot /host/ modprobe pci-pf-stub"}
{"level": "Level(-
4)", "ts":1616794345.5842702, "logger": "daemon.NodeConfigurator.loadModule", "msg": "co
mmands output", "output": ""}
{"level": "Level(-
4)", "ts":1616794345.5843055, "logger": "daemon.NodeConfigurator.loadModule", "msg": "ex
ecuting command", "cmd": "/usr/sbin/chroot /host/ modprobe vfio-pci"}
{"level": "Level(-
4)", "ts":1616794345.6090655, "logger": "daemon.NodeConfigurator.loadModule", "msg": "co
mmands output", "output": ""}
{"level": "Level(-
2)", "ts":1616794345.6091156, "logger": "daemon.NodeConfigurator", "msg": "device's
driver_override path", "path": "/sys/bus/pci/devices/0000:af:00.0/driver_override"}
{"level": "Level(-
2)", "ts":1616794345.6091807, "logger": "daemon.NodeConfigurator", "msg": "driver bind
path", "path": "/sys/bus/pci/drivers/pci-pf-stub/bind"}
{"level": "Level(-
2)", "ts":1616794345.7488534, "logger": "daemon.NodeConfigurator", "msg": "device's
driver_override path", "path": "/sys/bus/pci/devices/0000:b0:00.0/driver_override"}
{"level": "Level(-
2)", "ts":1616794345.748938, "logger": "daemon.NodeConfigurator", "msg": "driver bind
path", "path": "/sys/bus/pci/drivers/vfio-pci/bind"}
{"level": "Level(-
2)", "ts":1616794345.7492096, "logger": "daemon.NodeConfigurator", "msg": "device's
driver_override path", "path": "/sys/bus/pci/devices/0000:b0:00.1/driver_override"}
{"level": "Level(-
2)", "ts":1616794345.7492566, "logger": "daemon.NodeConfigurator", "msg": "driver bind
path", "path": "/sys/bus/pci/drivers/vfio-pci/bind"}
{"level": "Level(-
4)", "ts":1616794345.74968, "logger": "daemon.NodeConfigurator.applyConfig", "msg": "exec
uting command", "cmd": "/sriov_workdir/pf_bb_config ACC100 -c

```

```

/sriov_artifacts/0000:af:00.0.ini -p 0000:af:00.0"}
{"level":"Level(-
4)","ts":1616794346.5203931,"logger":"daemon.NodeConfigurator.applyConfig","msg":"co
mmands output","output":"Queue Groups: 0 5GUL, 0 5GDL, 4 4GUL, 4 4GDL\nNumber
of 5GUL engines 8\nConfiguration in VF mode\nPF ACC100 configuration
complete\nACC100 PF [0000:af:00.0] configuration complete!\n\n"}
{"level":"Level(-
4)","ts":1616794346.520459,"logger":"daemon.NodeConfigurator.enableMasterBus","msg"
:"executing command","cmd":"/usr/sbin/chroot /host/ setpci -v -s 0000:af:00.0
COMMAND"}
{"level":"Level(-
4)","ts":1616794346.5458736,"logger":"daemon.NodeConfigurator.enableMasterBus","ms
g":"commands output","output":"0000:af:00.0 @04 = 0142\n"}
{"level":"Level(-
4)","ts":1616794346.5459251,"logger":"daemon.NodeConfigurator.enableMasterBus","ms
g":"executing command","cmd":"/usr/sbin/chroot /host/ setpci -v -s 0000:af:00.0
COMMAND=0146"}
{"level":"Level(-
4)","ts":1616794346.5795262,"logger":"daemon.NodeConfigurator.enableMasterBus","ms
g":"commands output","output":"0000:af:00.0 @04 0146\n"}
{"level":"Level(-
2)","ts":1616794346.5795407,"logger":"daemon.NodeConfigurator.enableMasterBus","ms
g":"MasterBus set","pci":"0000:af:00.0","output":"0000:af:00.0 @04 0146\n"}
{"level":"Level(-
4)","ts":1616794346.6867144,"logger":"daemon.drainhelper.Run()","msg":"worker
function - end","performUncordon":true}
{"level":"Level(-
4)","ts":1616794346.6867719,"logger":"daemon.drainhelper.Run()","msg":"uncordoning
node"}
{"level":"Level(-
4)","ts":1616794346.6896322,"logger":"daemon.drainhelper.uncordon()","msg":"starting
uncordon attempts"}
{"level":"Level(-
2)","ts":1616794346.69735,"logger":"daemon.drainhelper.uncordon()","msg":"node
uncordoned"}
{"level":"Level(-
4)","ts":1616794346.6973662,"logger":"daemon.drainhelper.Run()","msg":"cancelling the
context to finish the leadership"}
{"level":"Level(-
4)","ts":1616794346.7029872,"logger":"daemon.drainhelper.Run()","msg":"stopped
leading"}
{"level":"Level(-
4)","ts":1616794346.7030034,"logger":"daemon.drainhelper","msg":"releasing the lock
(bug mitigation)"}
{"level":"Level(-
4)","ts":1616794346.8040674,"logger":"daemon.updateInventory","msg":"obtained
inventory","inv":{"sriovAccelerators":
[{"vendorID":"8086","deviceID":"0b32","pciAddress":"0000:20:00.0","driver":"","maxVirtualF
unctions":1,"virtualFunctions":[]},
{"vendorID":"8086","deviceID":"0d5c","pciAddress":"0000:af:00.0","driver":"pci-pf-
stub","maxVirtualFunctions":16,"virtualFunctions":
[{"pciAddress":"0000:b0:00.0","driver":"vfio-pci","deviceID":"0d5d"},
{"pciAddress":"0000:b0:00.1","driver":"vfio-pci","deviceID":"0d5d"}]}]}
{"level":"Level(-4)","ts":1616794346.9058325,"logger":"daemon","msg":"Update ignored,
generation unchanged"}

```

```
{
  "level": "Level(-2)",
  "ts": "1616794346.9065044",
  "logger": "daemon.Reconcile",
  "msg": "Reconciled",
  "namespace": "vran-acceleration-operators",
  "name": "pg-itengdvs02r.altera.com"
}
```

3. Check the FEC configuration of the card:

```
$ oc get sriovfecnodeconfig node1 -o yaml
```

### Example output

```
status:
  conditions:
  - lastTransitionTime: "2021-03-19T11:46:22Z"
    message: Configured successfully
    observedGeneration: 1
    reason: Succeeded
    status: "True"
    type: Configured
  inventory:
    sriovAccelerators:
    - deviceID: 0d5c 1
      driver: pci-pf-stub
      maxVirtualFunctions: 16
      pciAddress: 0000:af:00.0
      vendorID: "8086"
      virtualFunctions:
      - deviceID: 0d5d 2
        driver: vfio-pci
        pciAddress: 0000:b0:00.0
      - deviceID: 0d5d
        driver: vfio-pci
        pciAddress: 0000:b0:00.1
      - deviceID: 0d5d
        driver: vfio-pci
        pciAddress: 0000:b0:00.2
      - deviceID: 0d5d
        driver: vfio-pci
        pciAddress: 0000:b0:00.3
      - deviceID: 0d5d
        driver: vfio-pci
        pciAddress: 0000:b0:00.4
```

**1** The value **0d5c** is the **deviceID** physical function of the FEC device.

**2** The value **0d5d** is the **deviceID** virtual function of the FEC device.

### 16.4.5. Verifying application pod access and FPGA usage on OpenNESS

OpenNESS is an edge computing software toolkit that you can use to onboard and manage applications and network functions on any type of network.

To verify all OpenNESS features are working together, including SR-IOV binding, the device plugin, Wireless Base Band Device (bbdev) configuration, and SR-IOV (FEC) VF functionality inside a non-root pod, you can build an image and run a simple validation application for the device.

For more information, go to [openness.org](https://openness.org).

### Prerequisites

- Optional: Intel FPGA PAC N3000 card
- Node or nodes installed with the n3000-operator
- Node or nodes installed with the SR-IOV-FEC operator
- Real-Time kernel and huge pages configured with Performance Addon Operator
- Log in as a user with **cluster-admin** privileges

### Procedure

1. Create a namespace for the test by completing the following actions:
  - a. Define the **test-bbdev** namespace by creating a file named **test-bbdev-namespace.yaml** file as shown in the following example:

```
apiVersion: v1
kind: Namespace
metadata:
  name: test-bbdev
  labels:
    openshift.io/run-level: "1"
```

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

```
$ oc create -f test-bbdev-namespace.yaml
```

2. Create the following **Pod** specification, and then save the YAML in the **pod-test.yaml** file:

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-bbdev-sample-app
  namespace: test-bbdev 1
spec:
  containers:
  - securityContext:
    privileged: false
    capabilities:
      add:
      - IPC_LOCK
      - SYS_NICE
    name: bbdev-sample-app
    image: bbdev-sample-app:1.0 2
    command: [ "sudo", "/bin/bash", "-c", "--" ]
    runAsUser: 0 3
```

```

resources:
  requests:
    hugepages-1Gi: 4Gi 4
    memory: 1Gi
    cpu: "4" 5
    intel.com/intel_fec_5g: '1' 6
    #intel.com/intel_fec_acc100: '1'
    #intel.com/intel_fec_lte: '1'
  limits:
    memory: 4Gi
    cpu: "4"
    hugepages-1Gi: 4Gi
    intel.com/intel_fec_5g: '1'
    #intel.com/intel_fec_acc100: '1'
    #intel.com/intel_fec_lte: '1'

```

- 1 Specify the **namespace** you created in step 1.
- 2 This defines the test image containing the compiled DPDK.
- 3 Make the container execute internally as the root user.
- 4 Specify hugepage size **hugepages-1Gi** and the quantity of hugepages that will be allocated to the pod. Hugepages and isolated CPUs need to be configured using the Performance Addon Operator.
- 5 Specify the number of CPUs.
- 6 Testing of the N3000 5G FEC configuration is supported by **intel.com/intel\_fec\_5g**.



#### NOTE

To test the ACC100 configuration, uncomment **intel.com/intel\_fec\_acc100** by removing the # symbol. To test the N3000 4G/LTE configuration, uncomment **intel.com/intel\_fec\_lte** by removing the # symbol. Only one resource can be active at any time.

3. Create the pod:

```
$ oc apply -f pod-test.yaml
```

4. Check that the pod is created:

```
$ oc get pods -n test-bbdev
```

#### Example output

```

NAME                                READY    STATUS    RESTARTS    AGE
pod-bbdev-sample-app                1/1     Running   0           80s

```

5. Use a remote shell to log in to the **pod-bbdev-sample-app**:

```
$ oc rsh pod-bbdev-sample-app
```

■

### Example output

```
sh-4.4#
```

6. Print a list of environment variables:

```
sh-4.4# env
```

### Example output

```
N3000_CONTROLLER_MANAGER_METRICS_SERVICE_PORT_8443_TCP_ADDR=172.30.133.131
SRIOV_FEC_CONTROLLER_MANAGER_METRICS_SERVICE_PORT_8443_TCP_PROTOCOL=tcp
DPDK_VERSION=20.11
PCIDevice_INTEL_COM_INTEL_FEC_5G=0.0.0.0:1d.00.0 1
~/usr/bin/env
HOSTNAME=fec-pod
```

**1** This is the PCI address of the virtual function. Depending on the resource that you requested in the **pod-test.yaml** file, this can be any one of following three PCI addresses:

- PCIDevice\_INTEL\_COM\_INTEL\_FEC\_ACC100
- PCIDevice\_INTEL\_COM\_INTEL\_FEC\_5G
- PCIDevice\_INTEL\_COM\_INTEL\_FEC\_LTE

7. Change to the **test-bbdev** directory:

```
sh-4.4# cd test/test-bbdev/
```



### NOTE

The directory is in the pod and not on your local computer.

8. Check the CPUs that are assigned to the pod:

```
sh-4.4# export CPU=$(cat /sys/fs/cgroup/cpuset/cpuset.cpus)
sh-4.4# echo ${CPU}
```

This prints out the CPUs that are assigned to the **fec.pod**.

### Example output

```
24,25,64,65
```

9. Run the **test-bbdev** application to test the device:

```
sh-4.4# ./test-bbdev.py -e="-l ${CPU} -a ${PCIDevice_INTEL_COM_INTEL_FEC_5G}" -c
validation \ -n 64 -b 32 -l 1 -v ./test_vectors/"
```

### Example output

```
Executing: ../../build/app/dpdk-test-bbdev -l 24-25,64-65 0000:1d.00.0 -- -n 64 -l 1 -c
validation -v ./test_vectors/bbdev_null.data -b 32
EAL: Detected 80 lcore(s)
EAL: Detected 2 NUMA nodes
Option -w, --pci-whitelist is deprecated, use -a, --allow option instead
EAL: Multi-process socket /var/run/dpdk/rte/mp_socket
EAL: Selected IOVA mode 'VA'
EAL: Probing VFIO support...
EAL: VFIO support initialized
EAL: using IOMMU type 1 (Type 1)
EAL: Probe PCI driver: intel_fpga_5ngr_fec_vf (8086:d90) device: 0000:1d.00.0 (socket 1)
EAL: No legacy callbacks, legacy socket not created
```

```
=====
Starting Test Suite : BBdev Validation Tests
Test vector file = ldpc_dec_v7813.data
Device 0 queue 16 setup failed
Allocated all queues (id=16) at prio0 on dev0
Device 0 queue 32 setup failed
Allocated all queues (id=32) at prio1 on dev0
Device 0 queue 48 setup failed
Allocated all queues (id=48) at prio2 on dev0
Device 0 queue 64 setup failed
Allocated all queues (id=64) at prio3 on dev0
Device 0 queue 64 setup failed
All queues on dev 0 allocated: 64
+ ----- +
== test: validation
dev:0000:b0:00.0, burst size: 1, num ops: 1, op type: RTE_BBDEV_OP_LDPC_DEC
Operation latency:
    avg: 23092 cycles, 10.0838 us
    min: 23092 cycles, 10.0838 us
    max: 23092 cycles, 10.0838 us
TestCase [ 0 ] : validation_tc passed
+ ~~~~~ +
+ Test Suite Summary : BBdev Validation Tests
+ Tests Total :    1
+ Tests Skipped :  0
+ Tests Passed :  1 1
+ Tests Failed :  0
+ Tests Lasted :  177.67 ms
+ ~~~~~ +
```

**1** While some tests can be skipped, be sure that the vector tests pass.

## 16.5. ADDITIONAL RESOURCES



- [OpenNESS Operator for Intel® FPGA PAC N3000 \(Programming\)](#)
- [OpenNESS Operator for Wireless FEC Accelerators](#)