OpenShift Container Platform 4.15

Scalability and performance

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

OpenShift Container Platform 4.15 Scalability and performance

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

Legal Notice

Abstract

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

Table of Contents

CHAPTER 1. RECOMMENDED PERFORMANCE AND SCALABILITY PRACTICES

1.1. RECOMMENDED CONTROL PLANE PRACTICES

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

1.1.1. Recommended practices for scaling the cluster

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

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

When scaling up the cluster to higher node counts:

- Spread nodes across all of the available zones for higher availability.

- Scale up by no more than 25 to 50 machines at once.

- Consider creating new compute machine sets in each available zone with alternative instance types of similar size to help mitigate any periodic provider capacity constraints. For example, on AWS, use m5.large and m5d.large.

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

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

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

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

1.1.2. Control plane node sizing

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

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

- 1 image stream

- 1 build

- 5 deployments, with 2 pod replicas in a **sleep** state, mounting 4 secrets, 4 config maps, and 1 downward API volume each

- 5 services, each one pointing to the TCP/8080 and TCP/8443 ports of one of the previous deployments

- 1 route pointing to the first of the previous services

- 10 secrets containing 2048 random string characters

- 10 config maps containing 2048 random string characters

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

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

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

IMPORTANT

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

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

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

> **IMPORTANT**
>
> You can modify the control plane node size in a running OpenShift Container Platform 4.15 cluster for the following configurations only:
>
> - Clusters installed with a user-provisioned installation method.
>
> - AWS clusters installed with an installer-provisioned infrastructure installation method.
>
> - Clusters that use a control plane machine set to manage control plane machines.
>
> For all other configurations, you must estimate your total node count and use the suggested control plane node size during installation.

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

> **NOTE**
>
> In OpenShift Container Platform 4.15, half of a CPU core (500 millicore) is now reserved by the system by default compared to OpenShift Container Platform 3.11 and previous versions. The sizes are determined taking that into consideration.

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

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

> **NOTE**
>
> The procedure for clusters that use a control plane machine set is different from the procedure for clusters that do not use a control plane machine set.
>
> If you are uncertain about the state of the **ControlPlaneMachineSet** CR in your cluster, you can verify the CR status.

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

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

Prerequisites

- Your AWS cluster uses a control plane machine set.

Procedure

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

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

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

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

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

3. Save your changes.

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

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

Additional resources

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

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

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

Prerequisites

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

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

Procedure

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

2. Choose one control plane machine instance.

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

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

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

   d. Change the instance to a larger type, ensuring that the type is the same base as the previous selection, and apply changes. For example, you can change **m6i.xlarge** to **m6i.2xlarge** or **m6i.4xlarge**.

   e. Start the instance.

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

3. Repeat this process for each control plane machine.

Additional resources

- [Backing up etcd](#)

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

1.2. RECOMMENDED INFRASTRUCTURE PRACTICES

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

1.2.1. Infrastructure node sizing

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

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

In general, three infrastructure nodes are recommended per cluster.
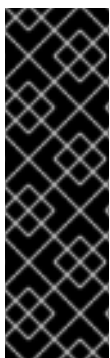
IMPORTANT

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

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

> **NOTE**
>
> In OpenShift Container Platform 4.15, half of a CPU core (500 millicore) is now reserved by the system by default compared to OpenShift Container Platform 3.11 and previous versions. This influences the stated sizing recommendations.

1.2.2. Scaling the Cluster Monitoring Operator

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

1.2.3. Prometheus database storage requirements

Red Hat performed various tests for different scale sizes.

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

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

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

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

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

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

Recommendations for OpenShift Container Platform

- Use at least two infrastructure (infra) nodes.

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

1.2.4. Configuring cluster monitoring

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

Procedure

To increase the storage capacity for Prometheus:

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

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

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

[2] [4] The storage class for your cluster.

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

[5] A typical value is **ALERTMANAGER_STORAGE_SIZE=20Gi**. Storage values can be a plain integer or a fixed-point integer using one of these suffixes: E, P, T, G, M, K. You can also

use the power-of-two equivalents: Ei, Pi, Ti, Gi, Mi, Ki.

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

3. Save the file.

4. Apply the changes by running:

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

1.2.5. Additional resources

- Infrastructure Nodes in OpenShift 4

- OpenShift Container Platform cluster maximums

- Creating infrastructure machine sets

1.3. RECOMMENDED ETCD PRACTICES

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

1.3.1. Recommended etcd practices

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

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

In terms of latency, run etcd on top of a block device that can write at least 50 IOPS of 8000 bytes long sequentially. That is, with a latency of 10ms, keep in mind that uses fdatasync to synchronize each write in the WAL. For heavy loaded clusters, sequential 500 IOPS of 8000 bytes (2 ms) are recommended. To measure those numbers, you can use a benchmarking tool, such as fio.

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

NOTE

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

The following hard drive practices provide optimal etcd performance:

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

- Prefer drives with low latency to support fast read and write operations.

- Prefer high-bandwidth writes for faster compactions and defragmentation.

- Prefer high-bandwidth reads for faster recovery from failures.

- Use solid state drives as a minimum selection. Prefer NVMe drives for production environments.

- Use server-grade hardware for increased reliability.

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

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

> **NOTE**
>
> Avoid using the Network File System (NFS) protocol or other network based file systems.

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

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

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

Prerequisites

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

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

Procedure

- Run fio and analyze the results:

  - If you use Podman, run this command:

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

- If you use Docker, run this command:

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

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

- **etcd_disk_wal_fsync_duration_seconds_bucket** metric reports the etcd's WAL fsync duration

- **etcd_disk_backend_commit_duration_seconds_bucket** metric reports the etcd backend commit latency duration

- **etcd_server_leader_changes_seen_total** metric reports the leader changes

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

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

Additional resources

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

- [etcd performance troubleshooting guide for OpenShift Container Platform](#)

1.3.2. Moving etcd to a different disk

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

The Machine Config Operator (MCO) is responsible for mounting a secondary disk for OpenShift Container Platform 4.15 container storage.

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

Prerequisites

- You have installed the OpenShift CLI (**oc**).

- You have access to the cluster with **cluster-admin** privileges.

- The **MachineConfigPool** must match **metadata.labels[machineconfiguration.openshift.io/role]**. This applies to a controller, worker, or a custom pool.

Procedure

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

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

   ```
   # lsblk
   ```

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

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

   ```
   apiVersion: machineconfiguration.openshift.io/v1
   kind: MachineConfig
   metadata:
     labels:
       machineconfiguration.openshift.io/role: master
     name: 98-var-lib-etcd
   spec:
     config:
       ignition:
         version: 3.4.0
       systemd:
         units:
         - contents: |
             [Unit]
             Description=Make File System on /dev/<new_disk_name>
             DefaultDependencies=no
             BindsTo=dev-<new_disk_name>.device
             After=dev-<new_disk_name>.device var.mount
             Before=systemd-fsck@dev-<new_disk_name>.service

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

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

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

```
        Options=defaults,prjquota

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

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

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

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

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

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

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

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

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

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

- The disk mounts to **/var/lib/etcd**.

- The content from **/sysroot/ostree/deploy/rhcos/var/lib/etcd** syncs to **/var/lib/etcd**.

- A restore of **SELinux** labels is forced for **/var/lib/etcd**.

- The old content is not removed.

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

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
 labels:
   machineconfiguration.openshift.io/role: master
 name: 98-var-lib-etcd
spec:
 config:
  ignition:
    version: 3.4.0
  systemd:
   units:
   - contents: |
      [Unit]
      Description=Mount /dev/<new_disk_name> to /var/lib/etcd
      Before=local-fs.target
      After=var.mount

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

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

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

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

Verification steps

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

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

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

Example output

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

Additional resources

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

1.3.3. Defragmenting etcd data

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

Monitor these key metrics:

- **etcd_server_quota_backend_bytes**, which is the current quota limit

- **etcd_mvcc_db_total_size_in_use_in_bytes**, which indicates the actual database usage after a history compaction

- **etcd_mvcc_db_total_size_in_bytes**, which shows the database size, including free space waiting for defragmentation

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

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

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

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

1.3.3.1. Automatic defragmentation

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

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

- etcd logs

- cluster-etcd-operator pod

- operator status error log

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

Example log output for successful defragmentation

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

Example log output for unsuccessful defragmentation

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

1.3.3.2. Manual defragmentation

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

- When etcd uses more than 50% of its available space for more than 10 minutes

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

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

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

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

Prerequisites

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

Procedure

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

a. Get the list of etcd pods:

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

Example output

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

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

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

Example output

```
Defaulting container name to etcdctl.
Use 'oc describe pod/etcd-ip-10-0-159-225.example.redhat.com -n openshift-etcd' to see all of the containers in this pod.
+--------------------------+------------------+---------+---------+-----------+------------+-----------+------------+--------------------+--------+
|         ENDPOINT         |        ID        | VERSION | DB SIZE | IS LEADER | IS LEARNER | RAFT TERM | RAFT INDEX | RAFT APPLIED INDEX | ERRORS |
+--------------------------+------------------+---------+---------+-----------+------------+-----------+------------+--------------------+--------+
|  https://10.0.191.37:2379 | 251cd44483d811c3 |   3.5.9 |  104 MB |    false |    false |     7 |    91624 |          91624 |      |
| https://10.0.159.225:2379 | 264c7c58ecbdabee |   3.5.9 |  104 MB |    false |    false |     7 |    91624 |          91624 |      |
| https://10.0.199.170:2379 | 9ac311f93915cc79 |   3.5.9 |  104 MB |    true |    false |     7 |    91624 |          91624 |      |
+--------------------------+------------------+---------+---------+-----------+------------+-----------+------------+--------------------+--------+
```

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

2. Defragment an etcd member.

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

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

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

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

c. Defragment the etcd member:

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

Example output

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

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

d. Verify that the database size was reduced:

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

Example output

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

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

e. Repeat these steps to connect to each of the other etcd members and defragment them. Always defragment the leader last.
Wait at least one minute between defragmentation actions to allow the etcd pod to recover. Until the etcd pod recovers, the etcd member will not respond.

3. If any **NOSPACE** alarms were triggered due to the space quota being exceeded, clear them.

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

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

Example output

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

b. Clear the alarms:

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

1.3.4. Setting tuning parameters for etcd

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

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

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

> **IMPORTANT**
>
> Tuning etcd latency tolerances is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.
>
> For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

1.3.4.1. Changing hardware speed tolerance

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

Prerequisites

- You have edited the cluster instance to enable Technology Preview features. For more information, see "Understanding feature gates".

Procedure

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

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

   Example output

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

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

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

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

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

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

3. Review the output:

Example output

> etcd.operator.openshift.io/cluster patched

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

Example output

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

4. Verify that the value was changed by entering the following command:

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

Example output

> Control Plane Hardware Speed: ""

5. Wait for etcd pods to roll out:

> oc get pods -n openshift-etcd -w

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

Example output

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

```
etcd-ci-ln-qkgs94t-72292-9clnd-master-0            0/4    Pending          0        0s
etcd-ci-ln-qkgs94t-72292-9clnd-master-0            0/4    Init:1/3         0        1s
etcd-ci-ln-qkgs94t-72292-9clnd-master-0            0/4    Init:2/3         0        2s
etcd-ci-ln-qkgs94t-72292-9clnd-master-0            0/4    PodInitializing  0        3s
etcd-ci-ln-qkgs94t-72292-9clnd-master-0            3/4    Running          0        4s
etcd-guard-ci-ln-qkgs94t-72292-9clnd-master-0       1/1    Running          0        26m
etcd-ci-ln-qkgs94t-72292-9clnd-master-0            3/4    Running          0        20s
etcd-ci-ln-qkgs94t-72292-9clnd-master-0            4/4    Running          0        20s
```

6. Enter the following command to review to the values:

   ```
   $ oc describe -n openshift-etcd pod/<ETCD_PODNAME> | grep -e HEARTBEAT_INTERVAL
   -e ELECTION_TIMEOUT
   ```
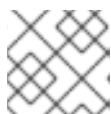
   > **NOTE**
   >
   > These values might not have changed from the default.

Additional resources

Understanding feature gates

CHAPTER 2. PLANNING YOUR ENVIRONMENT ACCORDING TO OBJECT MAXIMUMS

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

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

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

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

2.1. OPENSHIFT CONTAINER PLATFORM TESTED CLUSTER MAXIMUMS FOR MAJOR RELEASES

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

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

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

Maximum type	4.x tested maximum
Number of namespaces [5]	10,000
Number of builds	10,000 (Default pod RAM 512 Mi) - Source-to-Image (S2I) build strategy
Number of pods per namespace [6]	25,000
Number of routes and back ends per Ingress Controller	2,000 per router
Number of secrets	80,000
Number of config maps	90,000
Number of services [7]	10,000
Number of services per namespace	5,000
Number of back-ends per service	5,000
Number of deployments per namespace [6]	2,000
Number of build configs	12,000
Number of custom resource definitions (CRD)	1,024 [8]

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

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

3. This was tested on a cluster with 31 servers: 3 control planes, 2 infrastructure nodes, and 26 worker nodes. If you need 2,500 user pods, you need both a **hostPrefix** of **20**, which allocates a network large enough for each node to contain more than 2000 pods, and a custom kubelet config with **maxPods** set to **2500**. For more information, see Running 2500 pods per node on OCP 4.13.

4. The maximum tested pods per node is 2,500 for clusters using the **OVNKubernetes** network plugin. The maximum tested pods per node for the **OpenShiftSDN** network plugin is 500 pods.

5. 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.

6. There are several control loops in the system that must iterate over all objects in a given namespace as a reaction to some changes in state. Having a large number of objects of a given type in a single namespace can make those loops expensive and slow down processing given state changes. The limit assumes that the system has enough CPU, memory, and disk to satisfy the application requirements.

7. 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.

8. Tested on a cluster with 29 servers: 3 control planes, 2 infrastructure nodes, and 24 worker nodes. The cluster had 500 namespaces. OpenShift Container Platform has a limit of 1,024 total custom resource definitions (CRD), including those installed by OpenShift Container Platform, products integrating with OpenShift Container Platform and user-created CRDs. If there are more than 1,024 CRDs created, then there is a possibility that **oc** command requests might be throttled.

2.1.1. Example scenario

As an example, 500 worker nodes (m5.2xl) were tested, and are supported, using OpenShift Container Platform 4.15, the OVN-Kubernetes network plugin, and the following workload objects:

- 200 namespaces, in addition to the defaults

- 60 pods per node; 30 server and 30 client pods (30k total)

- 57 image streams/ns (11.4k total)

- 15 services/ns backed by the server pods (3k total)

- 15 routes/ns backed by the previous services (3k total)

- 20 secrets/ns (4k total)

- 10 config maps/ns (2k total)

- 6 network policies/ns, including deny-all, allow-from ingress and intra-namespace rules

- 57 builds/ns

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

- Number of pods per node

- Number of containers per pod

- Type of probes used (for example, liveness/readiness, exec/http)

- Number of network policies

- Number of projects, or namespaces

- Number of image streams per project

- Number of builds per project

- Number of services/endpoints and type

- Number of routes

- Number of shards

- Number of secrets

- Number of config maps

- Rate of API calls, or the cluster "churn", which is an estimation of how quickly things change in the cluster configuration.

  - Prometheus query for pod creation requests per second over 5 minute windows: **sum(irate(apiserver_request_count{resource="pods",verb="POST"}[5m]))**

  - Prometheus query for all API requests per second over 5 minute windows: **sum(irate(apiserver_request_count{}[5m]))**

- Cluster node resource consumption of CPU

- Cluster node resource consumption of memory

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

2.2.1. AWS cloud platform

Node	Flavor	vCPU	RAM(GiB)	Disk type	Disk size(GiB) /IOS	Count	Region
Control plane/etc d [1]	r5.4xlarge	16	128	gp3	220	3	us-west-2
Infra [2]	m5.12xlarge	48	192	gp3	100	3	us-west-2
Workload [3]	m5.4xlarge	16	64	gp3	500 [4]	1	us-west-2
Compute	m5.2xlarge	8	32	gp3	100	3/25/250 /500 [5]	us-west-2

1. gp3 disks with a baseline performance of 3000 IOPS and 125 MiB per second are used for control plane/etcd nodes because etcd is latency sensitive. gp3 volumes do not use burst performance.

2. Infra nodes are used to host Monitoring, Ingress, and Registry components to ensure they have enough resources to run at large scale.

3. Workload node is dedicated to run performance and scalability workload generators.

4. Larger disk size is used so that there is enough space to store the large amounts of data that is collected during the performance and scalability test run.

5. Cluster is scaled in iterations and performance and scalability tests are executed at the specified node counts.

2.2.2. IBM Power platform

Node	vCPU	RAM(GiB)	Disk type	Disk size(GiB)/IOS	Count
Control plane/etcd [1]	16	32	io1	120 / 10 IOPS per GiB	3
Infra [2]	16	64	gp2	120	2
Workload [3]	16	256	gp2	120 [4]	1
Compute	16	64	gp2	120	2 to 100 [5]

1. io1 disks with 120 / 10 IOPS per GiB are used for control plane/etcd nodes as etcd is I/O intensive and latency sensitive.

2. Infra nodes are used to host Monitoring, Ingress, and Registry components to ensure they have enough resources to run at large scale.

3. Workload node is dedicated to run performance and scalability workload generators.

4. Larger disk size is used so that there is enough space to store the large amounts of data that is collected during the performance and scalability test run.

5. Cluster is scaled in iterations.

2.2.3. IBM Z platform

Node	vCPU [4]	RAM(GiB)[5]	Disk type	Disk size(GiB)/IOS	Count
Control plane/etcd [1,2]	8	32	ds8k	300 / LCU 1	3
Compute [1,3]	8	32	ds8k	150 / LCU 2	4 nodes (scaled to 100/250/500 pods per node)

1. Nodes are distributed between two logical control units (LCUs) to optimize disk I/O load of the control plane/etcd nodes as etcd is I/O intensive and latency sensitive. Etcd I/O demand should not interfere with other workloads.

2. Four compute nodes are used for the tests running several iterations with 100/250/500 pods at the same time. First, idling pods were used to evaluate if pods can be instanced. Next, a network and CPU demanding client/server workload were used to evaluate the stability of the system under stress. Client and server pods were pairwise deployed and each pair was spread over two compute nodes.

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

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

5. Total physical memory used is 512 GiB.

2.3. HOW TO PLAN YOUR ENVIRONMENT ACCORDING TO TESTED CLUSTER MAXIMUMS
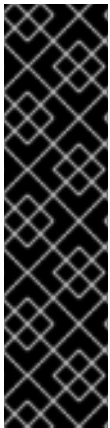
IMPORTANT

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

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

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

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

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

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

Example scenario

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

> 2200 / 500 = 4.4

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

> 2200 / 20 = 110

Where:

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

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

2.4. HOW TO PLAN YOUR ENVIRONMENT ACCORDING TO APPLICATION REQUIREMENTS

Consider an example application environment:

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

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

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

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

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

The application pods can access a service either by using environment variables or DNS. If using environment variables, for each active service the variables are injected by the kubelet when a pod is run on a node. A cluster-aware DNS server watches the Kubernetes API for new services and creates a set of DNS records for each one. If DNS is enabled throughout your cluster, then all pods should automatically be able to resolve services by their DNS name. Service discovery using DNS can be used in case you must go beyond 5000 services. When using environment variables for service discovery, the

argument list exceeds the allowed length after 5000 services in a namespace, then the pods and deployments will start failing. Disable the service links in the deployment's service specification file to overcome this:

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

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

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

- **<SERVICE_NAME>_SERVICE_HOST=<IP>**

- **<SERVICE_NAME>_SERVICE_PORT=<PORT>**

- **<SERVICE_NAME>_PORT=tcp://<IP>:<PORT>**

- **<SERVICE_NAME>_PORT_<PORT>_TCP=tcp://<IP>:<PORT>**

- **<SERVICE_NAME>_PORT_<PORT>_TCP_PROTO=tcp**

- **<SERVICE_NAME>_PORT_<PORT>_TCP_PORT=<PORT>**

- **<SERVICE_NAME>_PORT_<PORT>_TCP_ADDR=<ADDR>**

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

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

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

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

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

3.1. MANAGING CPU OVERCOMMITMENT

In a highly virtualized IBM Z® environment, you must carefully plan the infrastructure setup and sizing. One of the most important features of virtualization is the capability to do resource overcommitment, allocating more resources to the virtual machines than actually available at the hypervisor level. This is very workload dependent and there is no golden rule that can be applied to all setups.

Depending on your setup, consider these best practices regarding CPU overcommitment:

- At LPAR level (PR/SM hypervisor), avoid assigning all available physical cores (IFLs) to each LPAR. For example, with four physical IFLs available, you should not define three LPARs with four logical IFLs each.

- Check and understand LPAR shares and weights.

- An excessive number of virtual CPUs can adversely affect performance. Do not define more virtual processors to a guest than logical processors are defined to the LPAR.

- Configure the number of virtual processors per guest for peak workload, not more.

- Start small and monitor the workload. Increase the vCPU number incrementally if necessary.

- Not all workloads are suitable for high overcommitment ratios. If the workload is CPU intensive, you will probably not be able to achieve high ratios without performance problems. Workloads that are more I/O intensive can keep consistent performance even with high overcommitment ratios.

Additional resources

- z/VM Common Performance Problems and Solutions

- z/VM overcommitment considerations

- LPAR CPU management

3.2. DISABLE TRANSPARENT HUGE PAGES

Transparent Huge Pages (THP) attempt to automate most aspects of creating, managing, and using

huge pages. Since THP automatically manages the huge pages, this is not always handled optimally for all types of workloads. THP can lead to performance regressions, since many applications handle huge pages on their own. Therefore, consider disabling THP.

3.3. BOOST NETWORKING PERFORMANCE WITH RECEIVE FLOW STEERING

Receive Flow Steering (RFS) extends Receive Packet Steering (RPS) by further reducing network latency. RFS is technically based on RPS, and improves the efficiency of packet processing by increasing the CPU cache hit rate. RFS achieves this, and in addition considers queue length, by determining the most convenient CPU for computation so that cache hits are more likely to occur within the CPU. Thus, the CPU cache is invalidated less and requires fewer cycles to rebuild the cache. This can help reduce packet processing run time.

3.3.1. Use the Machine Config Operator (MCO) to activate RFS

Procedure

1. Copy the following MCO sample profile into a YAML file. For example, **enable-rfs.yaml**:

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
 labels:
   machineconfiguration.openshift.io/role: worker
 name: 50-enable-rfs
spec:
 config:
  ignition:
    version: 2.2.0
  storage:
   files:
   - contents:
      source: data:text/plain;charset=US-
ASCII,%23%20turn%20on%20Receive%20Flow%20Steering%20%28RFS%29%20for%20all
%20network%20interfaces%0ASUBSYSTEM%3D%3D%22net%22%2C%20ACTION%3D%
3D%22add%22%2C%20RUN%7Bprogram%7D%2B%3D%22/bin/bash%20-
c%20%27for%20x%20in%20/sys/%24DEVPATH/queues/rx-
%2A%3B%20do%20echo%208192%20%3E%20%24x/rps_flow_cnt%3B%20%20done%27
%22%0A
     filesystem: root
     mode: 0644
     path: /etc/udev/rules.d/70-persistent-net.rules
   - contents:
      source: data:text/plain;charset=US-
ASCII,%23%20define%20sock%20flow%20enbtried%20for%20%20Receive%20Flow%20Ste
ering%20%28RFS%29%0Anet.core.rps_sock_flow_entries%3D8192%0A
     filesystem: root
     mode: 0644
     path: /etc/sysctl.d/95-enable-rps.conf
```

2. Create the MCO profile:

```
$ oc create -f enable-rfs.yaml
```

3. Verify that an entry named **50-enable-rfs** is listed:

```
$ oc get mc
```

4. To deactivate, enter:

```
$ oc delete mc 50-enable-rfs
```

Additional resources

- OpenShift Container Platform on IBM Z®: Tune your network performance with RFS

- Configuring Receive Flow Steering (RFS)

- Scaling in the Linux Networking Stack

3.4. CHOOSE YOUR NETWORKING SETUP

The networking stack is one of the most important components for a Kubernetes-based product like OpenShift Container Platform. For IBM Z® setups, the networking setup depends on the hypervisor of your choice. Depending on the workload and the application, the best fit usually changes with the use case and the traffic pattern.

Depending on your setup, consider these best practices:

- Consider all options regarding networking devices to optimize your traffic pattern. Explore the advantages of OSA-Express, RoCE Express, HiperSockets, z/VM VSwitch, Linux Bridge (KVM), and others to decide which option leads to the greatest benefit for your setup.

- Always use the latest available NIC version. For example, OSA Express 7S 10 GbE shows great improvement compared to OSA Express 6S 10 GbE with transactional workload types, although both are 10 GbE adapters.

- Each virtual switch adds an additional layer of latency.

- The load balancer plays an important role for network communication outside the cluster. Consider using a production-grade hardware load balancer if this is critical for your application.

- OpenShift Container Platform SDN introduces flows and rules, which impact the networking performance. Make sure to consider pod affinities and placements, to benefit from the locality of services where communication is critical.

- Balance the trade-off between performance and functionality.

Additional resources

- OpenShift Container Platform on IBM Z® - Performance Experiences, Hints and Tips

- OpenShift Container Platform on IBM Z® Networking Performance

- Controlling pod placement on nodes using node affinity rules

3.5. ENSURE HIGH DISK PERFORMANCE WITH HYPERPAV ON Z/VM

DASD and ECKD devices are commonly used disk types in IBM Z® environments. In a typical OpenShift Container Platform setup in z/VM environments, DASD disks are commonly used to support the local storage for the nodes. You can set up HyperPAV alias devices to provide more throughput and overall better I/O performance for the DASD disks that support the z/VM guests.

Using HyperPAV for the local storage devices leads to a significant performance benefit. However, you must be aware that there is a trade-off between throughput and CPU costs.

3.5.1. Use the Machine Config Operator (MCO) to activate HyperPAV aliases in nodes using z/VM full-pack minidisks

For z/VM-based OpenShift Container Platform setups that use full-pack minidisks, you can leverage the advantage of MCO profiles by activating HyperPAV aliases in all of the nodes. You must add YAML configurations for both control plane and compute nodes.

Procedure

1. Copy the following MCO sample profile into a YAML file for the control plane node. For example, **05-master-kernelarg-hpav.yaml**:

   ```
   $ cat 05-master-kernelarg-hpav.yaml
   apiVersion: machineconfiguration.openshift.io/v1
   kind: MachineConfig
   metadata:
     labels:
       machineconfiguration.openshift.io/role: master
     name: 05-master-kernelarg-hpav
   spec:
     config:
       ignition:
         version: 3.1.0
     kernelArguments:
       - rd.dasd=800-805
   ```

2. Copy the following MCO sample profile into a YAML file for the compute node. For example, **05-worker-kernelarg-hpav.yaml**:

   ```
   $ cat 05-worker-kernelarg-hpav.yaml
   apiVersion: machineconfiguration.openshift.io/v1
   kind: MachineConfig
   metadata:
     labels:
       machineconfiguration.openshift.io/role: worker
     name: 05-worker-kernelarg-hpav
   spec:
     config:
       ignition:
         version: 3.1.0
     kernelArguments:
       - rd.dasd=800-805
   ```

   > **NOTE**
   >
   > You must modify the **rd.dasd** arguments to fit the device IDs.

3. Create the MCO profiles:

```
$ oc create -f 05-master-kernelarg-hpav.yaml
```

```
$ oc create -f 05-worker-kernelarg-hpav.yaml
```

4. To deactivate, enter:

```
$ oc delete -f 05-master-kernelarg-hpav.yaml
```

```
$ oc delete -f 05-worker-kernelarg-hpav.yaml
```

Additional resources

- Using HyperPAV for ECKD DASD

- Scaling HyperPAV alias devices on Linux guests on z/VM

3.6. RHEL KVM ON IBM Z HOST RECOMMENDATIONS

Optimizing a KVM virtual server environment strongly depends on the workloads of the virtual servers and on the available resources. The same action that enhances performance in one environment can have adverse effects in another. Finding the best balance for a particular setting can be a challenge and often involves experimentation.

The following section introduces some best practices when using OpenShift Container Platform with RHEL KVM on IBM Z® and IBM® LinuxONE environments.

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

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

The following example specifies **<iothreads>3</iothreads>** to configure three I/O threads, with consecutive decimal thread IDs 1, 2, and 3. The **iothread="2"** parameter specifies the driver element of the disk device to use the I/O thread with ID 2.

Sample I/O thread specification

```
...
<domain>
 <iothreads>3</iothreads>   1
   ...
    <devices>
     ...
        <disk type="block" device="disk">   2
<driver ... iothread="2"/>
    </disk>
     ...
    </devices>
   ...
</domain>
```

1 **1** The number of I/O threads.

2 The driver element of the disk device.

Threads can increase the performance of I/O operations for disk devices, but they also use memory and CPU resources. You can configure multiple devices to use the same thread. The best mapping of threads to devices depends on the available resources and the workload.

Start with a small number of I/O threads. Often, a single I/O thread for all disk devices is sufficient. Do not configure more threads than the number of virtual CPUs, and do not configure idle threads.

You can use the **virsh iothreadadd** command to add I/O threads with specific thread IDs to a running virtual server.

3.6.2. Avoid virtual SCSI devices

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

However, you might need SCSI-specific interfaces for:

- A LUN for a SCSI-attached tape drive on the host.

- A DVD ISO file on the host file system that is mounted on a virtual DVD drive.

3.6.3. Configure guest caching for disk

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

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

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

3.6.4. Exclude the memory balloon device

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

- Check the list of active profiles:

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

3.6.5. Tune the CPU migration algorithm of the host scheduler
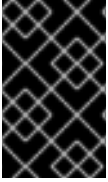
> **IMPORTANT**
>
> Do not change the scheduler settings unless you are an expert who understands the implications. Do not apply changes to production systems without testing them and confirming that they have the intended effect.

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

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

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

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

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

```
kernel.sched_migration_cost_ns=60000
```

3.6.6. Disable the cpuset cgroup controller

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

Procedure

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

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

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

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

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

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

   a. Stop all virtual machines.

   b. Run the following command:

      ```
      # systemctl restart libvirtd
      ```

   c. Restart the virtual machines.

This setting persists across host reboots.

3.6.7. Tune the polling period for idle virtual CPUs

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

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

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

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

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

Additional resources

- Linux on IBM Z® Performance Tuning for KVM

- Getting started with virtualization on IBM Z®

CHAPTER 4. USING THE NODE TUNING OPERATOR

Learn about the Node Tuning Operator and how you can use it to manage node-level tuning by orchestrating the tuned daemon.

4.1. ABOUT THE NODE TUNING OPERATOR

The Node Tuning Operator helps you manage node-level tuning by orchestrating the TuneD daemon and achieves low latency performance by using the Performance Profile controller. The majority of high-performance applications require some level of kernel tuning. The Node Tuning Operator provides a unified management interface to users of node-level sysctls and more flexibility to add custom tuning specified by user needs.

The Operator manages the containerized TuneD daemon for OpenShift Container Platform as a Kubernetes daemon set. It ensures the custom tuning specification is passed to all containerized TuneD daemons running in the cluster in the format that the daemons understand. The daemons run on all nodes in the cluster, one per node.

Node-level settings applied by the containerized TuneD daemon are rolled back on an event that triggers a profile change or when the containerized TuneD daemon is terminated gracefully by receiving and handling a termination signal.

The Node Tuning Operator uses the Performance Profile controller to implement automatic tuning to achieve low latency performance for OpenShift Container Platform applications.

The cluster administrator configures a performance profile to define node-level settings such as the following:

- Updating the kernel to kernel-rt.

- Choosing CPUs for housekeeping.

- Choosing CPUs for running workloads.

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

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

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

4.2. ACCESSING AN EXAMPLE NODE TUNING OPERATOR SPECIFICATION

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

Procedure

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

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

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

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

4.3. DEFAULT PROFILES SET ON A CLUSTER

The following are the default profiles set on a cluster.

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

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

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

4.4. VERIFYING THAT THE TUNED PROFILES ARE APPLIED

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

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

Example output

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

- **NAME**: Name of the Profile object. There is one Profile object per node and their names match.

- **TUNED**: Name of the desired TuneD profile to apply.

- **APPLIED**: **True** if the TuneD daemon applied the desired profile. (**True/False/Unknown**).

- **DEGRADED**: **True** if any errors were reported during application of the TuneD profile (**True/False/Unknown**).

- **AGE**: Time elapsed since the creation of Profile object.

The **ClusterOperator/node-tuning** object also contains useful information about the Operator and its node agents' health. For example, Operator misconfiguration is reported by **ClusterOperator/node-tuning** status messages.

To get status information about the **ClusterOperator/node-tuning** object, run the following command:

```
$ oc get co/node-tuning -n openshift-cluster-node-tuning-operator
```

Example output

```
NAME          VERSION  AVAILABLE  PROGRESSING  DEGRADED  SINCE  MESSAGE
node-tuning   4.15.1   True       False        True      60m    1/5 Profiles with bootcmdline conflict
```

If either the **ClusterOperator/node-tuning** or a profile object's status is **DEGRADED**, additional information is provided in the Operator or operand logs.

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

1   Optional.

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

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

4   An optional list.

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

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

7   Optional operand configuration.

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

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

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

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

1   Node or pod label name.

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

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

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

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

If **machineConfigLabels** is defined, machine config pool based matching is turned on for the given **recommend:** list item. **<mcLabels>** specifies the labels for a machine config. The machine config is

created automatically to apply host settings, such as kernel boot parameters, for the profile **<tuned_profile_name>**. This involves finding all machine config pools with machine config selector matching **<mcLabels>** and setting the profile **<tuned_profile_name>** on all nodes that are assigned the found machine config pools. To target nodes that have both master and worker roles, you must use the master role.

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

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

Example: Node or pod label based matching

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

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

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

Finally, the profile **openshift-node** has the lowest priority of **30**. It lacks the **<match>** section and, therefore, will always match. It acts as a profile catch-all to set **openshift-node** profile, if no other profile with higher priority matches on a given node.
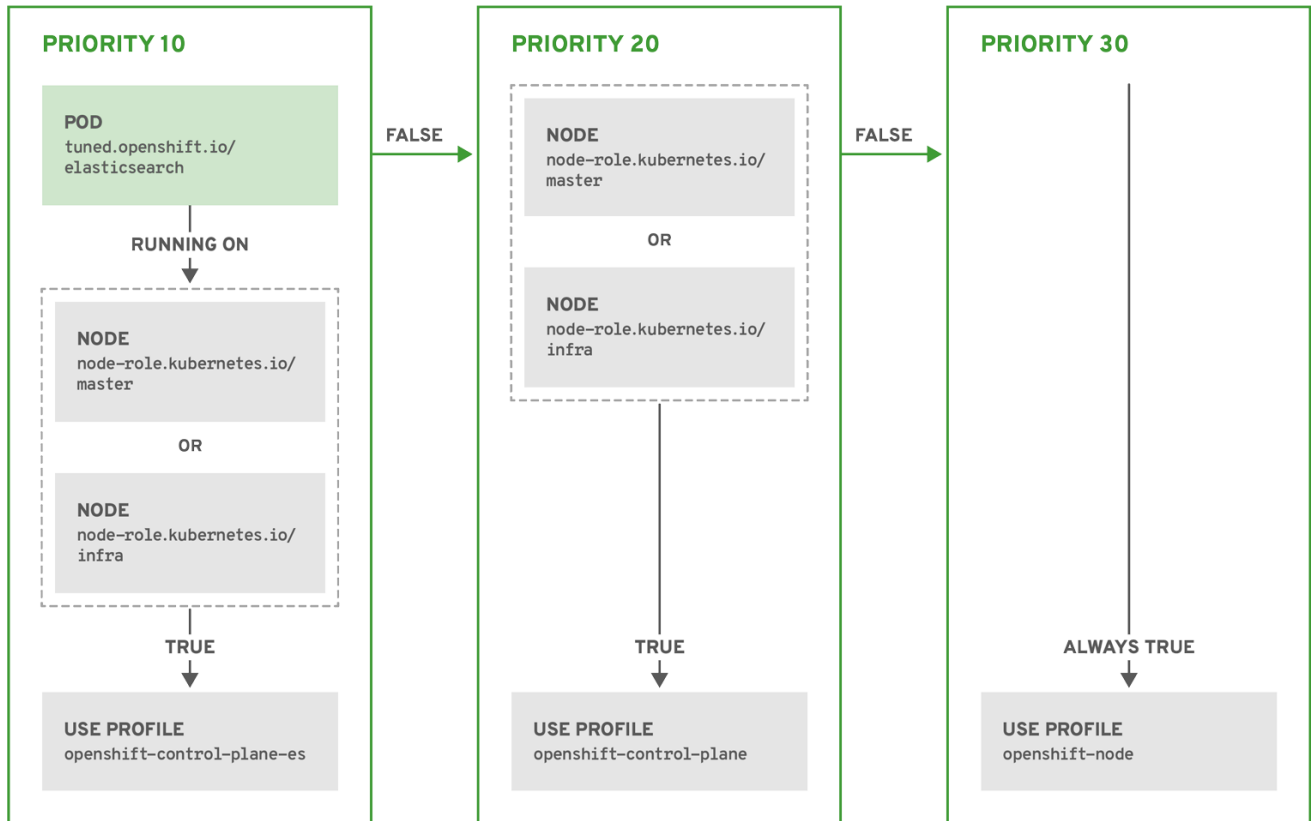
OPENSHIFT_10_0319

Example: Machine config pool based matching

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

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

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

Cloud provider-specific TuneD profiles

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

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

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

Example GCE Cloud provider profile

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

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

4.6. CUSTOM TUNING EXAMPLES

Using TuneD profiles from the default CR

The following CR applies custom node-level tuning for OpenShift Container Platform nodes with label **tuned.openshift.io/ingress-node-label** set to any value.

Example: custom tuning using the openshift-control-plane TuneD profile

```
apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
  name: ingress
  namespace: openshift-cluster-node-tuning-operator
spec:
  profile:
  - data: |
      [main]
      summary=A custom OpenShift ingress profile
      include=openshift-control-plane
```

```
    [sysctl]
    net.ipv4.ip_local_port_range="1024 65535"
    net.ipv4.tcp_tw_reuse=1
  name: openshift-ingress
recommend:
- match:
  - label: tuned.openshift.io/ingress-node-label
  priority: 10
  profile: openshift-ingress
```

> **IMPORTANT**
>
> Custom profile writers are strongly encouraged to include the default TuneD daemon
> profiles shipped within the default Tuned CR. The example above uses the default
> **openshift-control-plane** profile to accomplish this.

Using built-in TuneD profiles

Given the successful rollout of the NTO-managed daemon set, the TuneD operands all manage the
same version of the TuneD daemon. To list the built-in TuneD profiles supported by the daemon, query
any TuneD pod in the following way:

```
$ oc exec $tuned_pod -n openshift-cluster-node-tuning-operator -- find /usr/lib/tuned/ -name
tuned.conf -printf '%h\n' | sed 's|^.*/||'
```

You can use the profile names retrieved by this in your custom tuning specification.

Example: using built-in hpc-compute TuneD profile

```
apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
  name: openshift-node-hpc-compute
  namespace: openshift-cluster-node-tuning-operator
spec:
  profile:
  - data: |
      [main]
      summary=Custom OpenShift node profile for HPC compute workloads
      include=openshift-node,hpc-compute
    name: openshift-node-hpc-compute

  recommend:
  - match:
    - label: tuned.openshift.io/openshift-node-hpc-compute
    priority: 20
    profile: openshift-node-hpc-compute
```

In addition to the built-in **hpc-compute** profile, the example above includes the **openshift-node** TuneD
daemon profile shipped within the default Tuned CR to use OpenShift-specific tuning for compute
nodes.

Overriding host-level sysctls

Various kernel parameters can be changed at runtime by using **/run/sysctl.d/**, **/etc/sysctl.d/**, and **/etc/sysctl.conf** host configuration files. OpenShift Container Platform adds several host configuration files which set kernel parameters at runtime; for example, **net.ipv[4-6].**, **fs.inotify.**, and **vm.max_map_count**. These runtime parameters provide basic functional tuning for the system prior to the kubelet and the Operator start.

The Operator does not override these settings unless the **reapply_sysctl** option is set to **false**. Setting this option to **false** results in **TuneD** not applying the settings from the host configuration files after it applies its custom profile.

Example: overriding host-level sysctls

```
apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
  name: openshift-no-reapply-sysctl
  namespace: openshift-cluster-node-tuning-operator
spec:
  profile:
  - data: |
      [main]
      summary=Custom OpenShift profile
      include=openshift-node
      [sysctl]
      vm.max_map_count=>524288
    name: openshift-no-reapply-sysctl
  recommend:
  - match:
    - label: tuned.openshift.io/openshift-no-reapply-sysctl
    priority: 15
    profile: openshift-no-reapply-sysctl
    operand:
      tunedConfig:
        reapply_sysctl: false
```

4.7. SUPPORTED TUNED DAEMON PLUGINS

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

- audio

- cpu

- disk

- eeepc_she

- modules

- mounts

- net

- scheduler

- scsi_host

- selinux

- sysctl

- sysfs

- usb

- video

- vm

- bootloader

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

- script

- systemd

NOTE

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

Additional resources

- [Available TuneD Plugins](#)

- [Getting Started with TuneD](#)

4.8. CONFIGURING NODE TUNING IN A HOSTED CLUSTER

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

Procedure

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

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: tuned-1
  namespace: clusters
data:
  tuning: |
    apiVersion: tuned.openshift.io/v1
```

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

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

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

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

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

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

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

Verification

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

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

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

   Example output

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

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

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

   Example output

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

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

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

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

   Example output

   ```
   vm.dirty_ratio = 55
   ```

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

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

Procedure

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

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: tuned-hugepages
  namespace: clusters
data:
  tuning: |
    apiVersion: tuned.openshift.io/v1
    kind: Tuned
    metadata:
      name: hugepages
      namespace: openshift-cluster-node-tuning-operator
    spec:
      profile:
      - data: |
          [main]
          summary=Boot time configuration for hugepages
          include=openshift-node
          [bootloader]
          cmdline_openshift_node_hugepages=hugepagesz=2M hugepages=50
        name: openshift-node-hugepages
      recommend:
      - priority: 20
        profile: openshift-node-hugepages
```
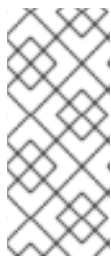
> **NOTE**
>
> The **.spec.recommend.match** field is intentionally left blank. In this case, this **Tuned** object is applied to all nodes in the node pool where this **ConfigMap** object is referenced. Group nodes with the same hardware configuration into the same node pool. Otherwise, TuneD operands can calculate conflicting kernel parameters for two or more nodes that share the same node pool.

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

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

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

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

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

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

```
apiVersion: hypershift.openshift.io/v1alpha1
kind: NodePool
metadata:
  name: hugepages-nodepool
  namespace: clusters
...
spec:
  management:
    ...
    upgradeType: InPlace
  ...
  tuningConfig:
  - name: tuned-hugepages
```
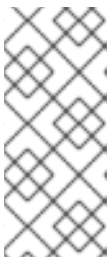
> **NOTE**
>
> To avoid the unnecessary re-creation of nodes when you apply the new **MachineConfig** objects, set **.spec.management.upgradeType** to **InPlace**. If you use the **Replace** upgrade type, nodes are fully deleted and new nodes can replace them when you apply the new kernel boot parameters that the TuneD operand calculated.

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

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

Verification

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

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

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

Example output

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

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

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

Example output

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

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

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

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

Example output

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

Additional resources

For more information about hosted control planes, see Hosted control planes.

CHAPTER 5. USING CPU MANAGER AND TOPOLOGY MANAGER

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

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

- Require as much CPU time as possible.

- Are sensitive to processor cache misses.

- Are low-latency network applications.

- Coordinate with other processes and benefit from sharing a single processor cache.

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

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

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

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

5.1. SETTING UP CPU MANAGER

Procedure

1. Optional: Label a node:

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

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

   ```
   # oc edit machineconfigpool worker
   ```

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

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

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

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

❶ Specify a policy:

- **none**. This policy explicitly enables the existing default CPU affinity scheme, providing no affinity beyond what the scheduler does automatically. This is the default policy.

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

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

5. Create the dynamic kubelet config:

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

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

6. Check for the merged kubelet config:

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

   Example output

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

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

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

   Example output

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

1   **cpuManagerPolicy** is defined when you create the **KubeletConfig** CR.

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

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

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

Example output

```yaml
apiVersion: v1
kind: Pod
metadata:
  generateName: cpumanager-
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
  - name: cpumanager
    image: gcr.io/google_containers/pause:3.2
    resources:
      requests:
        cpu: 1
        memory: "1G"
      limits:
        cpu: 1
        memory: "1G"
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
  nodeSelector:
    cpumanager: "true"
```

9. Create the pod:

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

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

```
# oc describe pod cpumanager
```

Example output

```
Name:         cpumanager-6cqz7
Namespace:    default
```

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

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

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

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

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

Example output

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

12. Check the allowed CPU list for the task:

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

Example output

```
 Cpus_allowed_list:    1
```

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

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

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

Example output

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

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

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

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

5.2. TOPOLOGY MANAGER POLICIES

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

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

none policy

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

best-effort policy

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

restricted policy

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

single-numa-node policy

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

5.3. SETTING UP TOPOLOGY MANAGER

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

Prerequisites

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

Procedure

To activate Topology Manager:

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

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

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

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

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

5.4. POD INTERACTIONS WITH TOPOLOGY MANAGER POLICIES

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

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

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

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

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

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

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

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

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

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

CHAPTER 6. SCHEDULING NUMA-AWARE WORKLOADS

Learn about NUMA-aware scheduling and how you can use it to deploy high performance workloads in an OpenShift Container Platform cluster.

The NUMA Resources Operator allows you to schedule high-performance workloads in the same NUMA zone. It deploys a node resources exporting agent that reports on available cluster node NUMA resources, and a secondary scheduler that manages the workloads.

6.1. ABOUT NUMA-AWARE SCHEDULING

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

NUMA architecture allows a CPU with multiple memory controllers to use any available memory across CPU complexes, regardless of where the memory is located. This allows for increased flexibility at the expense of performance. A CPU processing a workload using memory that is outside its NUMA zone is slower than a workload processed in a single NUMA zone. Also, for I/O-constrained workloads, the network interface on a distant NUMA zone slows down how quickly information can reach the application. High-performance workloads, such as telecommunications workloads, cannot operate to specification under these conditions. NUMA-aware scheduling aligns the requested cluster compute resources (CPUs, memory, devices) in the same NUMA zone to process latency-sensitive or high-performance workloads efficiently. NUMA-aware scheduling also improves pod density per compute node for greater resource efficiency.

By integrating the Node Tuning Operator's performance profile with NUMA-aware scheduling, you can further configure CPU affinity to optimize performance for latency-sensitive workloads.

The default OpenShift Container Platform pod scheduler scheduling logic considers the available resources of the entire compute node, not individual NUMA zones. If the most restrictive resource alignment is requested in the kubelet topology manager, error conditions can occur when admitting the pod to a node. Conversely, if the most restrictive resource alignment is not requested, the pod can be admitted to the node without proper resource alignment, leading to worse or unpredictable performance. For example, runaway pod creation with **Topology Affinity Error** statuses can occur when the pod scheduler makes suboptimal scheduling decisions for guaranteed pod workloads by not knowing if the pod's requested resources are available. Scheduling mismatch decisions can cause indefinite pod startup delays. Also, depending on the cluster state and resource allocation, poor pod scheduling decisions can cause extra load on the cluster because of failed startup attempts.

The NUMA Resources Operator deploys a custom NUMA resources secondary scheduler and other resources to mitigate against the shortcomings of the default OpenShift Container Platform pod scheduler. The following diagram provides a high-level overview of NUMA-aware pod scheduling.

Figure 6.1. NUMA-aware scheduling overview
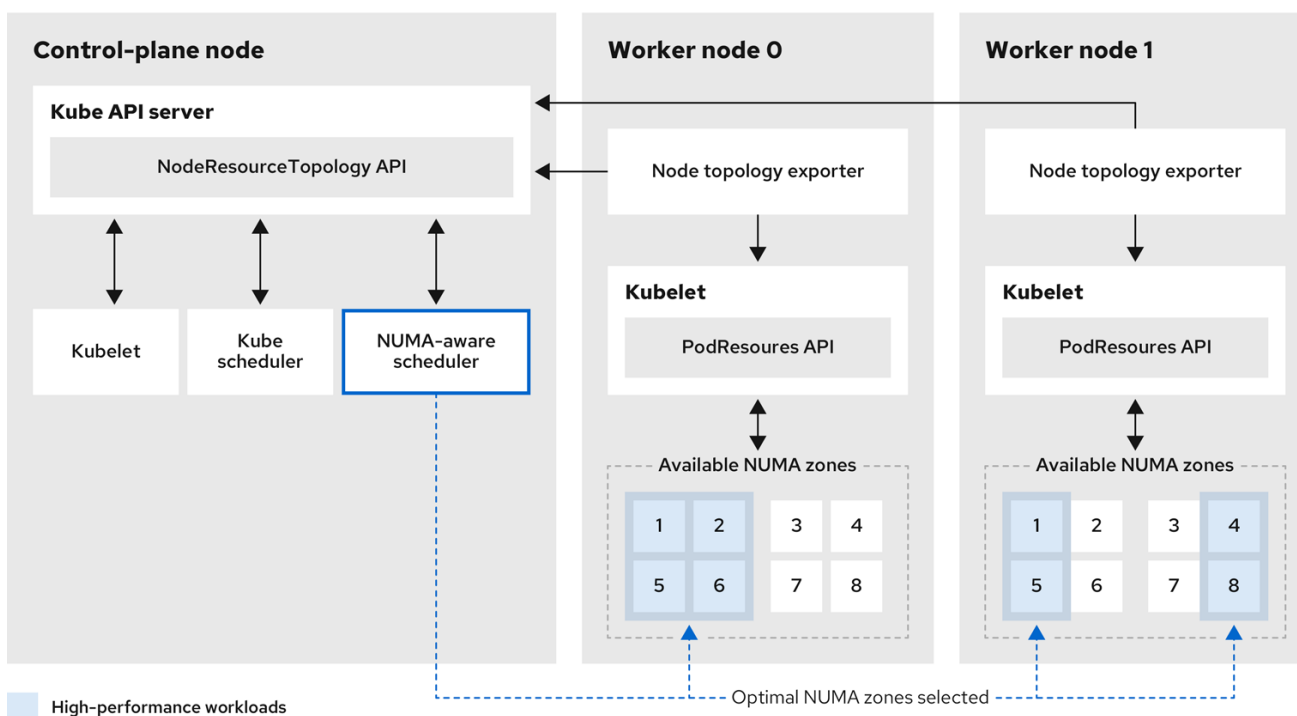
NodeResourceTopology API

The **NodeResourceTopology** API describes the available NUMA zone resources in each compute node.

NUMA-aware scheduler

The NUMA-aware secondary scheduler receives information about the available NUMA zones from the **NodeResourceTopology** API and schedules high-performance workloads on a node where it can be optimally processed.

Node topology exporter

The node topology exporter exposes the available NUMA zone resources for each compute node to the **NodeResourceTopology** API. The node topology exporter daemon tracks the resource allocation from the kubelet by using the **PodResources** API.

PodResources API

The **PodResources** API is local to each node and exposes the resource topology and available resources to the kubelet.

> **NOTE**
>
> The **List** endpoint of the **PodResources** API exposes exclusive CPUs allocated to a particular container. The API does not expose CPUs that belong to a shared pool.
>
> The **GetAllocatableResources** endpoint exposes allocatable resources available on a node.

Additional resources

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

6.2. INSTALLING THE NUMA RESOURCES OPERATOR

NUMA Resources Operator deploys resources that allow you to schedule NUMA-aware workloads and deployments. You can install the NUMA Resources Operator using the OpenShift Container Platform CLI or the web console.

6.2.1. Installing the NUMA Resources Operator using the CLI

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

Prerequisites

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

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

Procedure

1. Create a namespace for the NUMA Resources Operator:

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

      ```
      apiVersion: v1
      kind: Namespace
      metadata:
        name: openshift-numaresources
      ```

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

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

2. Create the Operator group for the NUMA Resources Operator:

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

      ```
      apiVersion: operators.coreos.com/v1
      kind: OperatorGroup
      metadata:
        name: numaresources-operator
        namespace: openshift-numaresources
      spec:
        targetNamespaces:
        - openshift-numaresources
      ```

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

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

3. Create the subscription for the NUMA Resources Operator:

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

      ```
      apiVersion: operators.coreos.com/v1alpha1
      kind: Subscription
      ```

```
    metadata:
      name: numaresources-operator
      namespace: openshift-numaresources
    spec:
      channel: "4.15"
      name: numaresources-operator
      source: redhat-operators
      sourceNamespace: openshift-marketplace
```

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

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

Verification

1. Verify that the installation succeeded by inspecting the CSV resource in the **openshift-numaresources** namespace. Run the following command:

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

Example output

```
NAME                        DISPLAY             VERSION   REPLACES   PHASE
numaresources-operator.v4.15.2   numaresources-operator   4.15.2               Succeeded
```

6.2.2. Installing the NUMA Resources Operator using the web console

As a cluster administrator, you can install the NUMA Resources Operator using the web console.

Procedure

1. Create a namespace for the NUMA Resources Operator:

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

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

2. Install the NUMA Resources Operator:

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

   b. Choose **NUMA Resources Operator** from the list of available Operators, and then click **Install**.

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

3. Optional: Verify that the NUMA Resources Operator installed successfully:

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

   b. Ensure that **NUMA Resources Operator** is listed in the **openshift-numaresources** namespace with a **Status** of **InstallSucceeded**.

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

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

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

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

6.3. SCHEDULING NUMA-AWARE WORKLOADS

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

6.3.1. Creating the NUMAResourcesOperator custom resource

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

Prerequisites

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

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

- Install the NUMA Resources Operator.

Procedure

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

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

   ```yaml
   apiVersion: nodetopology.openshift.io/v1
   kind: NUMAResourcesOperator
   metadata:
     name: numaresourcesoperator
   spec:
     nodeGroups:
     - machineConfigPoolSelector:
         matchLabels:
           pools.operator.machineconfiguration.openshift.io/worker: ""
   ```

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

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

Verification

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

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

Example output

```
NAME                AGE
numaresourcesoperator   10m
```

6.3.2. Deploying the NUMA-aware secondary pod scheduler

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

- Configure the performance profile.

- Deploy the NUMA-aware secondary scheduler.

Prerequisites

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

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

- Create the required machine config pool.

- Install the NUMA Resources Operator.

Procedure

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

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

   ```
   apiVersion: performance.openshift.io/v2
   kind: PerformanceProfile
   metadata:
     name: perfprof-nrop
   spec:
     cpu: ❶
       isolated: "4-51,56-103"
       reserved: "0,1,2,3,52,53,54,55"
     nodeSelector:
       node-role.kubernetes.io/worker: ""
     numa:
       topologyPolicy: single-numa-node
   ```

   ❶ The **cpu.isolated** and **cpu.reserved** specifications define ranges for isolated and reserved CPUs. Enter valid values for your CPU configuration. See the *Additional resources* section for more information about configuring a performance profile.

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

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

Example output

```
performanceprofile.performance.openshift.io/perfprof-nrop created
```

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

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

```
apiVersion: nodetopology.openshift.io/v1
kind: NUMAResourcesScheduler
metadata:
  name: numaresourcesscheduler
spec:
  imageSpec: "registry.redhat.io/openshift4/noderesourcetopology-scheduler-rhel9:v4.15"
  cacheResyncPeriod: "5s" 1
```

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

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

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

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

Verification

1. Verify that the performance profile was applied by running the following command:

```
$ oc describe performanceprofile <performance-profile-name>
```

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

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

Example output

```
NAME                                      READY  STATUS   RESTARTS  AGE
pod/numaresources-controller-manager-7575848485-bns4s  1/1    Running  0       13m
pod/numaresourcesoperator-worker-dvj4n              2/2    Running  0       16m
pod/numaresourcesoperator-worker-lcg4t              2/2    Running  0       16m
pod/secondary-scheduler-56994cf6cf-7qf4q            1/1    Running  0       16m
NAME                            DESIRED  CURRENT  READY  UP-TO-DATE
AVAILABLE   NODE SELECTOR            AGE
daemonset.apps/numaresourcesoperator-worker  2     2     2    2          2          node-
role.kubernetes.io/worker=   16m
NAME                            READY  UP-TO-DATE  AVAILABLE  AGE
deployment.apps/numaresources-controller-manager  1/1    1        1          13m
deployment.apps/secondary-scheduler              1/1    1        1          16m
NAME                            DESIRED  CURRENT  READY  AGE
replicaset.apps/numaresources-controller-manager-7575848485  1      1      1    13m
replicaset.apps/secondary-scheduler-56994cf6cf            1      1      1    16m
```

Additional resources

- [About the Performance Profile Creator](#).

6.3.3. Scheduling workloads with the NUMA-aware scheduler

You can schedule workloads with the NUMA-aware scheduler using **Deployment** CRs that specify the minimum required resources to process the workload.

The following example deployment uses NUMA-aware scheduling for a sample workload.

Prerequisites

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

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

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

Procedure

1. Get the name of the NUMA-aware scheduler that is deployed in the cluster by running the following command:

   ```
   $ oc get numaresourcesschedulers.nodetopology.openshift.io numaresourcesscheduler -o
   json | jq '.status.schedulerName'
   ```

   ### Example output

   ```
   topo-aware-scheduler
   ```

2. Create a **Deployment** CR that uses scheduler named **topo-aware-scheduler**, for example:

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

      ```
      apiVersion: apps/v1
      ```

```
kind: Deployment
metadata:
  name: numa-deployment-1
  namespace: openshift-numaresources
spec:
  replicas: 1
  selector:
    matchLabels:
      app: test
  template:
    metadata:
      labels:
        app: test
    spec:
      schedulerName: topo-aware-scheduler 1
      containers:
      - name: ctnr
        image: quay.io/openshifttest/hello-openshift:openshift
        imagePullPolicy: IfNotPresent
        resources:
          limits:
            memory: "100Mi"
            cpu: "10"
          requests:
            memory: "100Mi"
            cpu: "10"
      - name: ctnr2
        image: registry.access.redhat.com/rhel:latest
        imagePullPolicy: IfNotPresent
        command: ["/bin/sh", "-c"]
        args: [ "while true; do sleep 1h; done;" ]
        resources:
          limits:
            memory: "100Mi"
            cpu: "8"
          requests:
            memory: "100Mi"
            cpu: "8"
```

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

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

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

Verification

1. Verify that the deployment was successful:

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

Example output

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

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

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

Example output

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

> **NOTE**
>
> Deployments that request more resources than is available for scheduling will fail with a **MinimumReplicasUnavailable** error. The deployment succeeds when the required resources become available. Pods remain in the **Pending** state until the required resources are available.

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

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

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

   Example output

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

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

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

   Example output

   ```
   ...

   Zones:
    Costs:
   ```

```
              Name:   node-0
              Value:  10
              Name:   node-1
              Value:  21
          Name:     node-0
          Resources:
           Allocatable:  39
           Available:   21 ❶
           Capacity:    40
           Name:        cpu
           Allocatable:  6442450944
           Available:   6442450944
           Capacity:    6442450944
           Name:        hugepages-1Gi
           Allocatable:  134217728
           Available:   134217728
           Capacity:    134217728
           Name:        hugepages-2Mi
           Allocatable:  262415904768
           Available:   262206189568
           Capacity:    270146007040
           Name:        memory
        Type:         Node
```

❶ The **Available** capacity is reduced because of the resources that have been allocated to the guaranteed pod.

Resources consumed by guaranteed pods are subtracted from the available node resources listed under **noderesourcetopologies.topology.node.k8s.io**.

4. Resource allocations for pods with a **Best-effort** or **Burstable** quality of service (**qosClass**) are not reflected in the NUMA node resources under **noderesourcetopologies.topology.node.k8s.io**. If a pod's consumed resources are not reflected in the node resource calculation, verify that the pod has **qosClass** of **Guaranteed** and the CPU request is an integer value, not a decimal value. You can verify the that the pod has a **qosClass** of **Guaranteed** by running the following command:

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

Example output

```
Guaranteed
```

6.4. SCHEDULING NUMA-AWARE WORKLOADS WITH MANUAL PERFORMANCE SETTINGS

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

6.4.1. Creating the NUMAResourcesOperator custom resource with manual performance settings

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

Prerequisites

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

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

- Install the NUMA Resources Operator.

Procedure

1. Optional: Create the **MachineConfigPool** custom resource that enables custom kubelet configurations for worker nodes:
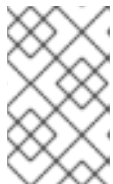
   > **NOTE**
   >
   > By default, OpenShift Container Platform creates a **MachineConfigPool** resource for worker nodes in the cluster. You can create a custom **MachineConfigPool** resource if required.

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

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

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

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

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

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

   ```yaml
   apiVersion: nodetopology.openshift.io/v1
   kind: NUMAResourcesOperator
   ```

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

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

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

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

Verification

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

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

Example output

```
NAME                 AGE
numaresourcesoperator   10m
```

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

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

- Configure the pod admittance policy for the required machine profile

- Create the required machine config pool

- Deploy the NUMA-aware secondary scheduler

Prerequisites

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

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

- Install the NUMA Resources Operator.

Procedure

1. Create the **KubeletConfig** custom resource that configures the pod admittance policy for the machine profile:

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

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: cnf-worker-tuning
spec:
  machineConfigPoolSelector:
    matchLabels:
      cnf-worker-tuning: enabled
  kubeletConfig:
    cpuManagerPolicy: "static" 1
    cpuManagerReconcilePeriod: "5s"
    reservedSystemCPUs: "0,1"
    memoryManagerPolicy: "Static" 2
    evictionHard:
      memory.available: "100Mi"
    reservedMemory:
      - numaNode: 0
        limits:
          memory: "1124Mi"
    systemReserved:
      memory: "512Mi"
    topologyManagerPolicy: "single-numa-node" 3
    topologyManagerScope: "pod"
```

1 For **cpuManagerPolicy**, **static** must use a lowercase **s**.

2 For **memoryManagerPolicy**, **Static** must use an uppercase **S**.

3 **topologyManagerPolicy** must be set to **single-numa-node**.

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

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

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

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

```
apiVersion: nodetopology.openshift.io/v1
kind: NUMAResourcesScheduler
metadata:
  name: numaresourcesscheduler
spec:
  imageSpec: "registry.redhat.io/openshift4/noderesourcetopology-scheduler-container-rhel8:v4.15"
  cacheResyncPeriod: "5s" 1
```

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

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

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

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

Verification

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

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

Example output

```
NAME                                                READY   STATUS    RESTARTS   AGE
pod/numaresources-controller-manager-7575848485-bns4s   1/1     Running   0          13m
pod/numaresourcesoperator-worker-dvj4n                  2/2     Running   0          16m
pod/numaresourcesoperator-worker-lcg4t                  2/2     Running   0          16m
pod/secondary-scheduler-56994cf6cf-7qf4q                1/1     Running   0          16m
NAME                                    DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR             AGE
daemonset.apps/numaresourcesoperator-worker   2     2     2     2     2     node-role.kubernetes.io/worker=   16m
NAME                                    READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/numaresources-controller-manager   1/1   1     1     13m
deployment.apps/secondary-scheduler                1/1   1     1     16m
NAME                                    DESIRED   CURRENT   READY   AGE
replicaset.apps/numaresources-controller-manager-7575848485   1     1     1     13m
replicaset.apps/secondary-scheduler-56994cf6cf               1     1     1     16m
```

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

You can schedule workloads with the NUMA-aware scheduler using **Deployment** CRs that specify the minimum required resources to process the workload.

The following example deployment uses NUMA-aware scheduling for a sample workload.

Prerequisites

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

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

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

Procedure

1. Get the name of the NUMA-aware scheduler that is deployed in the cluster by running the following command:

   ```
   $ oc get numaresourcesschedulers.nodetopology.openshift.io numaresourcesscheduler -o
   json | jq '.status.schedulerName'
   ```

   Example output

   ```
   topo-aware-scheduler
   ```

2. Create a **Deployment** CR that uses scheduler named **topo-aware-scheduler**, for example:

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

   ```yaml
   apiVersion: apps/v1
   kind: Deployment
   metadata:
     name: numa-deployment-1
     namespace: openshift-numaresources
   spec:
     replicas: 1
     selector:
       matchLabels:
         app: test
     template:
       metadata:
         labels:
           app: test
       spec:
         schedulerName: topo-aware-scheduler 1
         containers:
         - name: ctnr
           image: quay.io/openshifttest/hello-openshift:openshift
           imagePullPolicy: IfNotPresent
           resources:
             limits:
               memory: "100Mi"
               cpu: "10"
             requests:
               memory: "100Mi"
               cpu: "10"
         - name: ctnr2
           image: registry.access.redhat.com/rhel:latest
           imagePullPolicy: IfNotPresent
           command: ["/bin/sh", "-c"]
           args: [ "while true; do sleep 1h; done;" ]
           resources:
             limits:
               memory: "100Mi"
   ```

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

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

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

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

Verification

1. Verify that the deployment was successful:

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

Example output

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

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

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

Example output

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

> **NOTE**
>
> Deployments that request more resources than is available for scheduling will fail with a **MinimumReplicasUnavailable** error. The deployment succeeds when the required resources become available. Pods remain in the **Pending** state until the required resources are available.

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

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

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

Example output

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

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

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

Example output

```
...

Zones:
  Costs:
    Name:   node-0
    Value:  10
    Name:   node-1
    Value:  21
  Name:     node-0
  Resources:
    Allocatable:  39
    Available:    21 ❶
    Capacity:     40
    Name:         cpu
    Allocatable:  6442450944
    Available:    6442450944
    Capacity:     6442450944
    Name:         hugepages-1Gi
    Allocatable:  134217728
    Available:    134217728
    Capacity:     134217728
    Name:         hugepages-2Mi
    Allocatable:  262415904768
    Available:    262206189568
    Capacity:     270146007040
    Name:         memory
  Type:         Node
```

❶ The **Available** capacity is reduced because of the resources that have been allocated to the guaranteed pod.

Resources consumed by guaranteed pods are subtracted from the available node resources listed under **noderesourcetopologies.topology.node.k8s.io**.

4. Resource allocations for pods with a **Best-effort** or **Burstable** quality of service (**qosClass**) are not reflected in the NUMA node resources under **noderesourcetopologies.topology.node.k8s.io**. If a pod's consumed resources are not

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

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

Example output

```
Guaranteed
```

6.5. OPTIONAL: CONFIGURING POLLING OPERATIONS FOR NUMA RESOURCES UPDATES

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

The configuration options are the following:

- **infoRefreshMode**: Determines the trigger condition for polling the kubelet. The NUMA Resources Operator reports the resulting information to the API server.

- **infoRefreshPeriod**: Determines the duration between polling updates.

- **podsFingerprinting**: Determines if point-in-time information for the current set of pods running on a node is exposed in polling updates.

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

Prerequisites

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

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

- Install the NUMA Resources Operator.

Procedure

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

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

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

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

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

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

Verification

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

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

Example output

```
...

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

...
```

6.6. TROUBLESHOOTING NUMA-AWARE SCHEDULING

To troubleshoot common problems with NUMA-aware pod scheduling, perform the following steps.

Prerequisites

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

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

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

Procedure

1. Verify that the **noderesourcetopologies** CRD is deployed in the cluster by running the following command:

```
$ oc get crd | grep noderesourcetopologies
```

Example output

```
NAME                                          CREATED AT
noderesourcetopologies.topology.node.k8s.io          2022-01-18T08:28:06Z
```

2. Check that the NUMA-aware scheduler name matches the name specified in your NUMA-aware workloads by running the following command:

```
$ oc get numaresourcesschedulers.nodetopology.openshift.io numaresourcesscheduler -o
json | jq '.status.schedulerName'
```

Example output

```
topo-aware-scheduler
```

3. Verify that NUMA-aware schedulable nodes have the **noderesourcetopologies** CR applied to them. Run the following command:

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

Example output

```
NAME                AGE
compute-0.example.com   17h
compute-1.example.com   17h
```
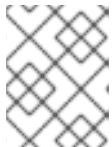
> **NOTE**
>
> The number of nodes should equal the number of worker nodes that are configured by the machine config pool (**mcp**) worker definition.

4. Verify the NUMA zone granularity for all schedulable nodes by running the following command:

```
$ oc get noderesourcetopologies.topology.node.k8s.io -o yaml
```

Example output

```
apiVersion: v1
items:
- apiVersion: topology.node.k8s.io/v1
  kind: NodeResourceTopology
  metadata:
    annotations:
      k8stopoawareschedwg/rte-update: periodic
    creationTimestamp: "2022-06-16T08:55:38Z"
    generation: 63760
    name: worker-0
    resourceVersion: "8450223"
    uid: 8b77be46-08c0-4074-927b-d49361471590
```

```
topologyPolicies:
- SingleNUMANodeContainerLevel
zones:
- costs:
  - name: node-0
    value: 10
  - name: node-1
    value: 21
  name: node-0
  resources:
  - allocatable: "38"
    available: "38"
    capacity: "40"
    name: cpu
  - allocatable: "134217728"
    available: "134217728"
    capacity: "134217728"
    name: hugepages-2Mi
  - allocatable: "262352048128"
    available: "262352048128"
    capacity: "270107316224"
    name: memory
  - allocatable: "6442450944"
    available: "6442450944"
    capacity: "6442450944"
    name: hugepages-1Gi
  type: Node
- costs:
  - name: node-0
    value: 21
  - name: node-1
    value: 10
  name: node-1
  resources:
  - allocatable: "268435456"
    available: "268435456"
    capacity: "268435456"
    name: hugepages-2Mi
  - allocatable: "269231067136"
    available: "269231067136"
    capacity: "270573244416"
    name: memory
  - allocatable: "40"
    available: "40"
    capacity: "40"
    name: cpu
  - allocatable: "1073741824"
    available: "1073741824"
    capacity: "1073741824"
    name: hugepages-1Gi
  type: Node
- apiVersion: topology.node.k8s.io/v1
  kind: NodeResourceTopology
  metadata:
    annotations:
      k8stopoawareschedwg/rte-update: periodic
```

```
    creationTimestamp: "2022-06-16T08:55:37Z"
    generation: 62061
    name: worker-1
    resourceVersion: "8450129"
    uid: e8659390-6f8d-4e67-9a51-1ea34bba1cc3
topologyPolicies:
- SingleNUMANodeContainerLevel
zones: 1
- costs:
  - name: node-0
    value: 10
  - name: node-1
    value: 21
  name: node-0
  resources: 2
  - allocatable: "38"
    available: "38"
    capacity: "40"
    name: cpu
  - allocatable: "6442450944"
    available: "6442450944"
    capacity: "6442450944"
    name: hugepages-1Gi
  - allocatable: "134217728"
    available: "134217728"
    capacity: "134217728"
    name: hugepages-2Mi
  - allocatable: "262391033856"
    available: "262391033856"
    capacity: "270146301952"
    name: memory
  type: Node
- costs:
  - name: node-0
    value: 21
  - name: node-1
    value: 10
  name: node-1
  resources:
  - allocatable: "40"
    available: "40"
    capacity: "40"
    name: cpu
  - allocatable: "1073741824"
    available: "1073741824"
    capacity: "1073741824"
    name: hugepages-1Gi
  - allocatable: "268435456"
    available: "268435456"
    capacity: "268435456"
    name: hugepages-2Mi
  - allocatable: "269192085504"
    available: "269192085504"
    capacity: "270534262784"
    name: memory
  type: Node
```

```
kind: List
metadata:
  resourceVersion: ""
  selfLink: ""
```

1   Each stanza under **zones** describes the resources for a single NUMA zone.

2   **resources** describes the current state of the NUMA zone resources. Check that resources listed under **items.zones.resources.available** correspond to the exclusive NUMA zone resources allocated to each guaranteed pod.

6.6.1. Checking the NUMA-aware scheduler logs

Troubleshoot problems with the NUMA-aware scheduler by reviewing the logs. If required, you can increase the scheduler log level by modifying the **spec.logLevel** field of the **NUMAResourcesScheduler** resource. Acceptable values are **Normal**, **Debug**, and **Trace**, with **Trace** being the most verbose option.

> **NOTE**
>
> To change the log level of the secondary scheduler, delete the running scheduler resource and re-deploy it with the changed log level. The scheduler is unavailable for scheduling new workloads during this downtime.

Prerequisites

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

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

Procedure

1. Delete the currently running **NUMAResourcesScheduler** resource:

   a. Get the active **NUMAResourcesScheduler** by running the following command:

      ```
      $ oc get NUMAResourcesScheduler
      ```

      Example output

      ```
      NAME                  AGE
      numaresourcesscheduler   90m
      ```

   b. Delete the secondary scheduler resource by running the following command:

      ```
      $ oc delete NUMAResourcesScheduler numaresourcesscheduler
      ```

      Example output

      ```
      numaresourcesscheduler.nodetopology.openshift.io "numaresourcesscheduler" deleted
      ```

2. Save the following YAML in the file **nro-scheduler-debug.yaml**. This example changes the log level to **Debug**:

```
apiVersion: nodetopology.openshift.io/v1
kind: NUMAResourcesScheduler
metadata:
  name: numaresourcesscheduler
spec:
  imageSpec: "registry.redhat.io/openshift4/noderesourcetopology-scheduler-container-rhel8:v4.15"
  logLevel: Debug
```

3. Create the updated **Debug** logging **NUMAResourcesScheduler** resource by running the following command:

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

Example output

```
numaresourcesscheduler.nodetopology.openshift.io/numaresourcesscheduler created
```

Verification steps

1. Check that the NUMA-aware scheduler was successfully deployed:

   a. Run the following command to check that the CRD is created succesfully:

   ```
   $ oc get crd | grep numaresourcesschedulers
   ```

   Example output

   ```
   NAME                                        CREATED AT
   numaresourcesschedulers.nodetopology.openshift.io          2022-02-25T11:57:03Z
   ```

   b. Check that the new custom scheduler is available by running the following command:

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

   Example output

   ```
   NAME                AGE
   numaresourcesscheduler   3h26m
   ```

2. Check that the logs for the scheduler shows the increased log level:

   a. Get the list of pods running in the **openshift-numaresources** namespace by running the following command:

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

   Example output

```
NAME                                       READY   STATUS    RESTARTS   AGE
numaresources-controller-manager-d87d79587-76mrm   1/1     Running   0          46h
numaresourcesoperator-worker-5wm2k                 2/2     Running   0          45h
numaresourcesoperator-worker-pb75c                 2/2     Running   0          45h
secondary-scheduler-7976c4d466-qm4sc               1/1     Running   0          21m
```

b. Get the logs for the secondary scheduler pod by running the following command:

```
$ oc logs secondary-scheduler-7976c4d466-qm4sc -n openshift-numaresources
```

Example output

```
...
I0223 11:04:55.614788       1 reflector.go:535] k8s.io/client-go/informers/factory.go:134:
Watch close - *v1.Namespace total 11 items received
I0223 11:04:56.609114       1 reflector.go:535] k8s.io/client-go/informers/factory.go:134:
Watch close - *v1.ReplicationController total 10 items received
I0223 11:05:22.626818       1 reflector.go:535] k8s.io/client-go/informers/factory.go:134:
Watch close - *v1.StorageClass total 7 items received
I0223 11:05:31.610356       1 reflector.go:535] k8s.io/client-go/informers/factory.go:134:
Watch close - *v1.PodDisruptionBudget total 7 items received
I0223 11:05:31.713032       1 eventhandlers.go:186] "Add event for scheduled pod"
pod="openshift-marketplace/certified-operators-thtvq"
I0223 11:05:53.461016       1 eventhandlers.go:244] "Delete event for scheduled pod"
pod="openshift-marketplace/certified-operators-thtvq"
```

6.6.2. Troubleshooting the resource topology exporter

Troubleshoot **noderesourcetopologies** objects where unexpected results are occurring by inspecting the corresponding **resource-topology-exporter** logs.

> **NOTE**
>
> It is recommended that NUMA resource topology exporter instances in the cluster are named for nodes they refer to. For example, a worker node with the name **worker** should have a corresponding **noderesourcetopologies** object called **worker**.

Prerequisites

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

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

Procedure

1. Get the daemonsets managed by the NUMA Resources Operator. Each daemonset has a corresponding **nodeGroup** in the **NUMAResourcesOperator** CR. Run the following command:

```
$ oc get numaresourcesoperators.nodetopology.openshift.io numaresourcesoperator -o
jsonpath="{.status.daemonsets[0]}"
```

Example output

```
{"name":"numaresourcesoperator-worker","namespace":"openshift-numaresources"}
```

2. Get the label for the daemonset of interest using the value for **name** from the previous step:

```
$ oc get ds -n openshift-numaresources numaresourcesoperator-worker -o jsonpath="
{.spec.selector.matchLabels}"
```

Example output

```
{"name":"resource-topology"}
```

3. Get the pods using the **resource-topology** label by running the following command:

```
$ oc get pods -n openshift-numaresources -l name=resource-topology -o wide
```

Example output

```
NAME                           READY  STATUS   RESTARTS  AGE   IP          NODE
numaresourcesoperator-worker-5wm2k  2/2    Running  0         2d1h  10.135.0.64
compute-0.example.com
numaresourcesoperator-worker-pb75c  2/2    Running  0         2d1h  10.132.2.33
compute-1.example.com
```

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

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

Example output

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

6.6.3. Correcting a missing resource topology exporter config map

If you install the NUMA Resources Operator in a cluster with misconfigured cluster settings, in some circumstances, the Operator is shown as active but the logs of the resource topology exporter (RTE) daemon set pods show that the configuration for the RTE is missing, for example:

> Info: couldn't find configuration in "/etc/resource-topology-exporter/config.yaml"

This log message indicates that the **kubeletconfig** with the required configuration was not properly applied in the cluster, resulting in a missing RTE **configmap**. For example, the following cluster is missing a **numaresourcesoperator-worker configmap** custom resource (CR):

> $ oc get configmap

Example output

```
NAME                     DATA   AGE
0e2a6bd3.openshift-kni.io    0      6d21h
kube-root-ca.crt             1      6d21h
openshift-service-ca.crt     1      6d21h
topo-aware-scheduler-config  1      6d18h
```

In a correctly configured cluster, **oc get configmap** also returns a **numaresourcesoperator-worker configmap** CR.

Prerequisites

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

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

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

Procedure

1. Compare the values for **spec.machineConfigPoolSelector.matchLabels** in **kubeletconfig** and **metadata.labels** in the **MachineConfigPool** (**mcp**) worker CR using the following commands:

   a. Check the **kubeletconfig** labels by running the following command:

      > $ oc get kubeletconfig -o yaml

      ### Example output

      ```
      machineConfigPoolSelector:
        matchLabels:
          cnf-worker-tuning: enabled
      ```

   b. Check the **mcp** labels by running the following command:

      > $ oc get mcp worker -o yaml

      ### Example output

```
labels:
  machineconfiguration.openshift.io/mco-built-in: ""
  pools.operator.machineconfiguration.openshift.io/worker: ""
```

The **cnf-worker-tuning: enabled** label is not present in the **MachineConfigPool** object.

2. Edit the **MachineConfigPool** CR to include the missing label, for example:

```
$ oc edit mcp worker -o yaml
```

Example output

```
labels:
  machineconfiguration.openshift.io/mco-built-in: ""
  pools.operator.machineconfiguration.openshift.io/worker: ""
  cnf-worker-tuning: enabled
```

3. Apply the label changes and wait for the cluster to apply the updated configuration. Run the following command:

Verification

- Check that the missing **numaresourcesoperator-worker configmap** CR is applied:

```
$ oc get configmap
```

Example output

```
NAME                          DATA   AGE
0e2a6bd3.openshift-kni.io     0      6d21h
kube-root-ca.crt              1      6d21h
numaresourcesoperator-worker  1      5m
openshift-service-ca.crt      1      6d21h
topo-aware-scheduler-config   1      6d18h
```

6.6.4. Collecting NUMA Resources Operator data

You can use the **oc adm must-gather** CLI command to collect information about your cluster, including features and objects associated with the NUMA Resources Operator.

Prerequisites

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

- You have installed the OpenShift CLI (**oc**).

Procedure

- To collect NUMA Resources Operator data with **must-gather**, you must specify the NUMA Resources Operator **must-gather** image.

```
$ oc adm must-gather --image=registry.redhat.io/numaresources-must-
gather/numaresources-must-gather-rhel9:4.15
```

CHAPTER 7. SCALABILITY AND PERFORMANCE OPTIMIZATION

7.1. OPTIMIZING STORAGE

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

7.1.1. Available persistent storage options

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

Table 7.1. Available storage options

Storage type	Description	Examples
Block	- Presented to the operating system (OS) as a block device - Suitable for applications that need full control of storage and operate at a low level on files bypassing the file system - Also referred to as a Storage Area Network (SAN) - Non-shareable, which means that only one client at a time can mount an endpoint of this type	AWS EBS and VMware vSphere support dynamic persistent volume (PV) provisioning natively in OpenShift Container Platform.
File	- Presented to the OS as a file system export to be mounted - Also referred to as Network Attached Storage (NAS) - Concurrency, latency, file locking mechanisms, and other capabilities vary widely between protocols, implementations, vendors, and scales.	RHEL NFS, NetApp NFS [1], and Vendor NFS
Object	- Accessible through a REST API endpoint - Configurable for use in the OpenShift image registry - Applications must build their drivers into the application and/or container.	AWS S3

1. NetApp NFS supports dynamic PV provisioning when using the Trident plugin.

7.1.2. Recommended configurable storage technology

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

Table 7.2. Recommended and configurable storage technology

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

[1] **ReadOnlyMany**

[2] **ReadWriteMany**

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

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

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

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

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

NOTE

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

7.1.2.1. Specific application storage recommendations

IMPORTANT

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

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

7.1.2.1.1. Registry

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

- The storage technology does not have to support RWX access mode.

- The storage technology must ensure read-after-write consistency.

- The preferred storage technology is object storage followed by block storage.

- File storage is not recommended for OpenShift image registry cluster deployment with production workloads.

7.1.2.1.2. Scaled registry

In a scaled/HA OpenShift image registry cluster deployment:

- The storage technology must support RWX access mode.

- The storage technology must ensure read-after-write consistency.

- The preferred storage technology is object storage.

- Red Hat OpenShift Data Foundation (ODF), Amazon Simple Storage Service (Amazon S3), Google Cloud Storage (GCS), Microsoft Azure Blob Storage, and OpenStack Swift are supported.

- Object storage should be S3 or Swift compliant.

- For non-cloud platforms, such as vSphere and bare metal installations, the only configurable technology is file storage.

- Block storage is not configurable.

7.1.2.1.3. Metrics

In an OpenShift Container Platform hosted metrics cluster deployment:

- The preferred storage technology is block storage.

- Object storage is not configurable.

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

7.1.2.1.4. Logging

In an OpenShift Container Platform hosted logging cluster deployment:

- Loki Operator:

    - The preferred storage technology is S3 compatible Object storage.

    - Block storage is not configurable.

- OpenShift Elasticsearch Operator:

    - The preferred storage technology is block storage.

    - Object storage is not supported.

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

7.1.2.1.5. Applications

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

- Storage technologies that support dynamic PV provisioning have low mount time latencies, and are not tied to nodes to support a healthy cluster.

- Application developers are responsible for knowing and understanding the storage requirements for their application, and how it works with the provided storage to ensure that issues do not occur when an application scales or interacts with the storage layer.

7.1.2.2. Other specific application storage recommendations

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

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

- Databases: Databases (RDBMSs, NoSQL DBs, etc.) tend to perform best with dedicated block storage.

- The etcd database must have enough storage and adequate performance capacity to enable a large cluster. Information about monitoring and benchmarking tools to establish ample storage and a high-performance environment is described in *Recommended etcd practices*.

7.1.3. Data storage management

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

Table 7.3. Main directories for storing OpenShift Container Platform data

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

7.1.4. Optimizing storage performance for Microsoft Azure

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

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

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

7.1.5. Additional resources

- Configuring the Elasticsearch log store

7.2. OPTIMIZING ROUTING

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

7.2.1. Baseline Ingress Controller (router) performance

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

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

- HTTP keep-alive/close mode

- Route type

- TLS session resumption client support

- Number of concurrent connections per target route

- Number of target routes

- Back end server page size

- Underlying infrastructure (network/SDN solution, CPU, and so on)

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

In HTTP keep-alive mode scenarios:

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

In HTTP close (no keep-alive) scenarios:

Encryption	LoadBalancerService	HostNetwork
none	5719	8273
edge	2729	4069
passthrough	4121	5344
re-encrypt	2320	2941

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Verification

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

7.2.3. Configuring HAProxy reload interval

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

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

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

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

Procedure

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

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

7.3. OPTIMIZING NETWORKING

The OpenShift SDN uses OpenvSwitch, virtual extensible LAN (VXLAN) tunnels, OpenFlow rules, and iptables. This network can be tuned by using jumbo frames, 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 plugins that implement different routing techniques, such as border gateway protocol (BGP).

- Use VXLAN-offload capable network adapters. VXLAN-offload moves the packet checksum calculation and associated CPU overhead off of the system CPU and onto dedicated hardware on the network adapter. This frees up CPU cycles for use by pods and applications, and allows users to utilize the full bandwidth of their network infrastructure.

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

7.3.1. Optimizing the MTU for your network

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

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

The OpenShift SDN network plugin overlay MTU must be less than the NIC MTU by 50 bytes at a minimum. This accounts for the SDN overlay header. So, on a normal ethernet network, this should be set to **1450**. On a jumbo frame ethernet network, this should be set to **8950**. These values should be set automatically by the Cluster Network Operator based on the NIC's configured MTU. Therefore, cluster administrators do not typically update these values. Amazon Web Services (AWS) and bare-metal environments support jumbo frame ethernet networks. This setting will help throughput, especially with transmission control protocol (TCP).

NOTE

OpenShift SDN CNI is deprecated as of OpenShift Container Platform 4.14. As of OpenShift Container Platform 4.15, the network plugin is not an option for new installations. In a subsequent future release, the OpenShift SDN network plugin is planned to be removed and no longer supported. Red Hat will provide bug fixes and support for this feature until it is removed, but this feature will no longer receive enhancements. As an alternative to OpenShift SDN CNI, you can use OVN Kubernetes CNI instead.

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

NOTE

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

7.3.2. Recommended practices for installing large scale clusters

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

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

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

7.3.3. Impact of IPsec

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

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

7.3.4. Additional resources

- Modifying advanced network configuration parameters

- Configuration parameters for the OVN-Kubernetes network plugin

- Configuration parameters for the OpenShift SDN network plugin

- Improving cluster stability in high latency environments using worker latency profiles

7.4. OPTIMIZING CPU USAGE WITH MOUNT NAMESPACE ENCAPSULATION

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

> **IMPORTANT**
>
> Mount namespace encapsulation is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.
>
> For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

7.4.1. Encapsulating mount namespaces

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

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

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

The following diagrams illustrate a Kubernetes installation before and after encapsulation. Both scenarios show example containers which have mount propagation settings of bidirectional, host-to-container, and none.
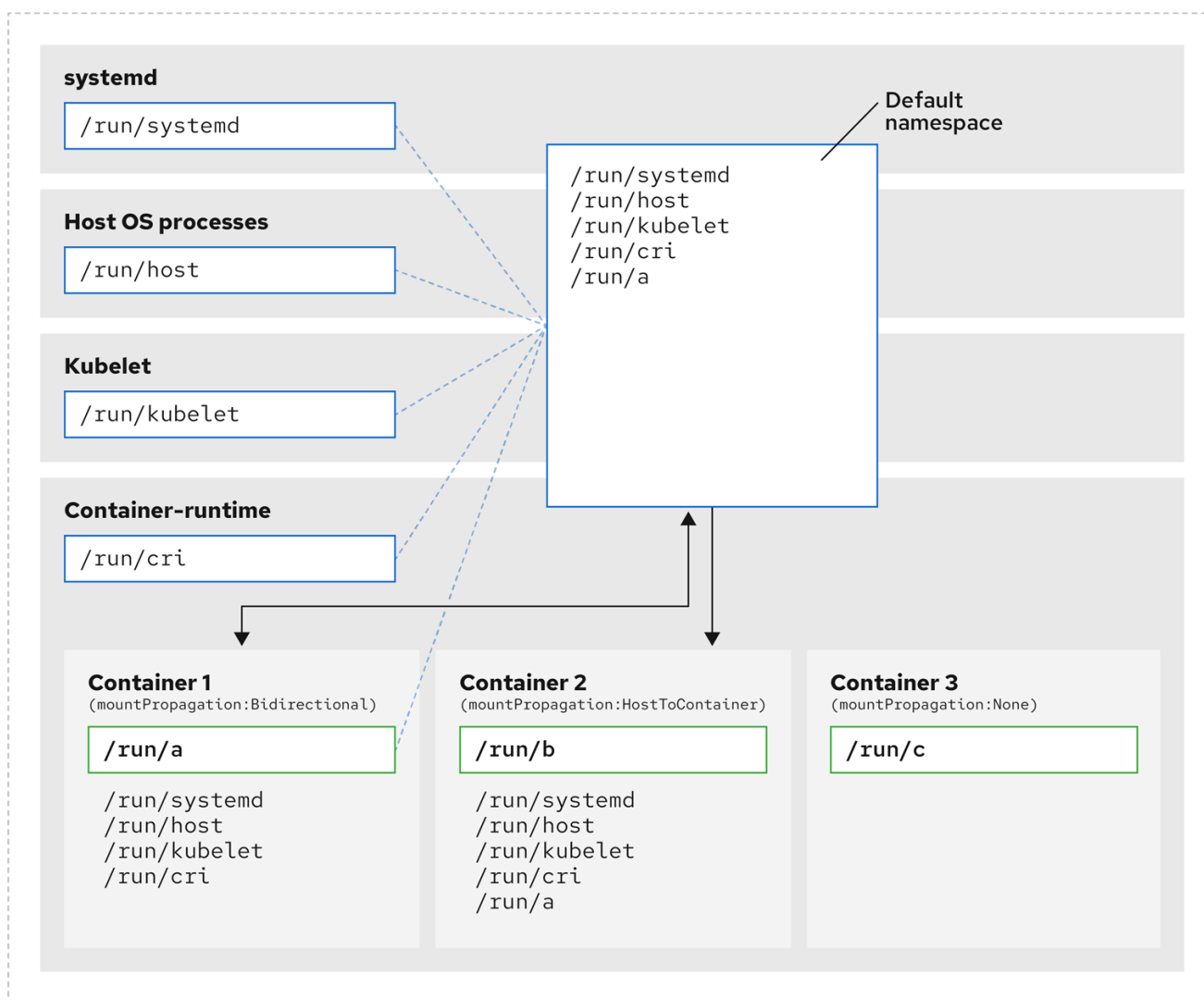
Here we see systemd, host operating system processes, kubelet, and the container runtime sharing a single mount namespace.

- systemd, host operating system processes, kubelet, and the container runtime each have access to and visibility of all mount points.

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

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

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

The following diagram illustrates the system state after encapsulation.
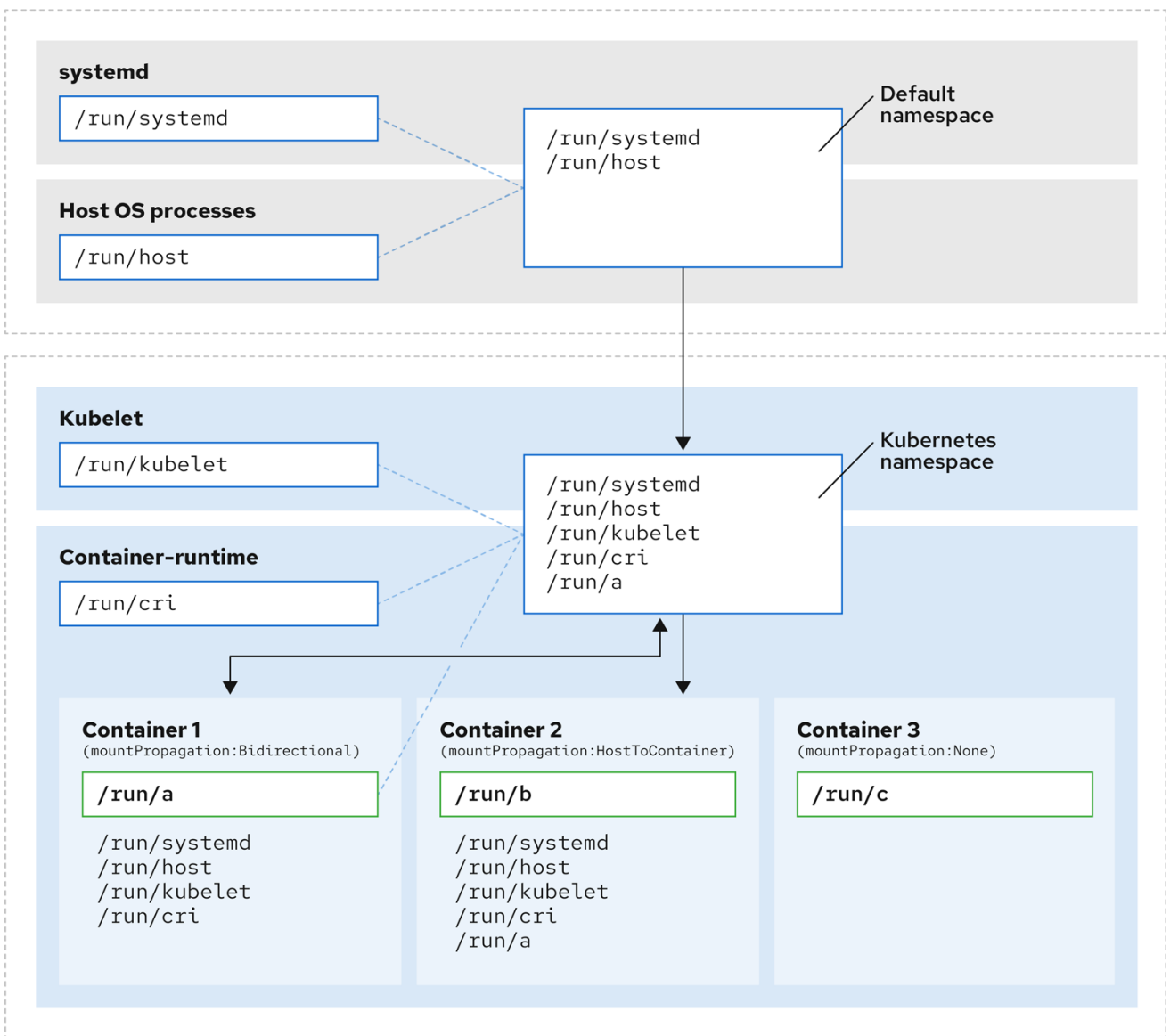
- The main systemd process is no longer devoted to unnecessary scanning of Kubernetes-specific mount points. It only monitors systemd-specific and host mount points.

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

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

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

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

7.4.2. Configuring mount namespace encapsulation

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

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

Prerequisites

- You have installed the OpenShift CLI (**oc**).

- You have logged in as a user with **cluster-admin** privileges.

Procedure

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

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

```
    ignition:
     version: 3.2.0
    systemd:
     units:
     - enabled: true
       name: kubens.service
```

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

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

Example output

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

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

```
$ oc get mcp
```

Example output

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

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

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

Example output

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

Verification

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

1. Open a debug shell to the cluster host:

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

2. Open a **chroot** session:

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

3. Check the systemd mount namespace:

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

   Example output

   ```
   mnt:[4026531953]
   ```

4. Check kubelet mount namespace:

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

   Example output

   ```
   mnt:[4026531840]
   ```

5. Check the CRI-O mount namespace:

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

   Example output

   ```
   mnt:[4026531840]
   ```

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

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

7.4.3. Inspecting encapsulated namespaces

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

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

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

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

Prerequisites

- You have installed the OpenShift CLI (**oc**).

- You have logged in as a user with **cluster-admin** privileges.

- You have configured SSH access to the cluster host.

Procedure

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

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

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

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

   Example output

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

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

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

   Example output

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

7.4.4. Running additional services in the encapsulated namespace

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

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

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

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

7.4.5. Additional resources

- What are namespaces

- Manage containers in namespaces by using nsenter

- MachineConfig

CHAPTER 8. MANAGING BARE METAL HOSTS

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

8.1. ABOUT BARE METAL HOSTS AND NODES

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

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

8.2. MAINTAINING BARE METAL HOSTS

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

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

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

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

2. Scales down the related compute machine set

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

Additional resources

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

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

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

Prerequisites

- Install an RHCOS cluster on bare metal.

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

Procedure

1. In the web console, navigate to **Compute → Bare Metal Hosts**.

2. Select **Add Host → New with Dialog**.

3. Specify a unique name for the new bare metal host.

4. Set the **Boot MAC address**.

5. Set the **Baseboard Management Console (BMC) Address**.

6. Enter the user credentials for the host's baseboard management controller (BMC).

7. Select to power on the host after creation, and select **Create**.

8. Scale up the number of replicas to match the number of available bare metal hosts. Navigate to **Compute → MachineSets**, and increase the number of machine replicas in the cluster by selecting **Edit Machine count** from the **Actions** drop-down menu.
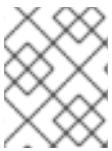
> **NOTE**
>
> You can also manage the number of bare metal nodes using the **oc scale** command and the appropriate bare metal compute machine set.

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

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

Prerequisites

- Install a RHCOS compute machine on bare metal infrastructure for use in the cluster.

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

- Create a **Secret** CR for the bare metal host.

Procedure

1. In the web console, navigate to **Compute → Bare Metal Hosts**.

2. Select **Add Host → New from YAML**.

3. Copy and paste the below YAML, modifying the relevant fields with the details of your host:

```yaml
apiVersion: metal3.io/v1alpha1
kind: BareMetalHost
metadata:
  name: <bare_metal_host_name>
spec:
  online: true
  bmc:
```

```
    address: <bmc_address>
    credentialsName: <secret_credentials_name>  1
    disableCertificateVerification: True  2
  bootMACAddress: <host_boot_mac_address>
```

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

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

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

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

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

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

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

Prerequisites

- Install RHCOS bare metal compute machines for use in the cluster, and create corresponding **BareMetalHost** objects.

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

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

Procedure

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

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

   Wait for the new scaled machines to start.

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

8.2.4. Removing bare metal hosts from the provisioner node

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

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

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

Prerequisites

- Install RHCOS bare metal compute machines for use in the cluster and create corresponding **BareMetalHost** objects.

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

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

Procedure

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

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

   Wait for the new machines to start.

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

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

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

Additional resources

- [Expanding the cluster](#)

- [MachineHealthChecks on bare metal](#)

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

IMPORTANT

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

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

9.1. ABOUT BARE-METAL EVENTS

IMPORTANT

The Bare Metal Event Relay Operator is deprecated. The ability to monitor bare-metal hosts by using the Bare Metal Event Relay Operator will be removed in a future OpenShift Container Platform release.

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

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

Hardware-related events published through Redfish includes:

- Breaches of temperature limits

- Server status

- Fan status

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

NOTE

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

9.2. HOW BARE-METAL EVENTS WORK

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

AMQP mechanism. The latency of the messaging service is between 10 to 20 milliseconds.

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

9.2.1. Bare Metal Event Relay data flow

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

Figure 9.1. Bare Metal Event Relay data flow
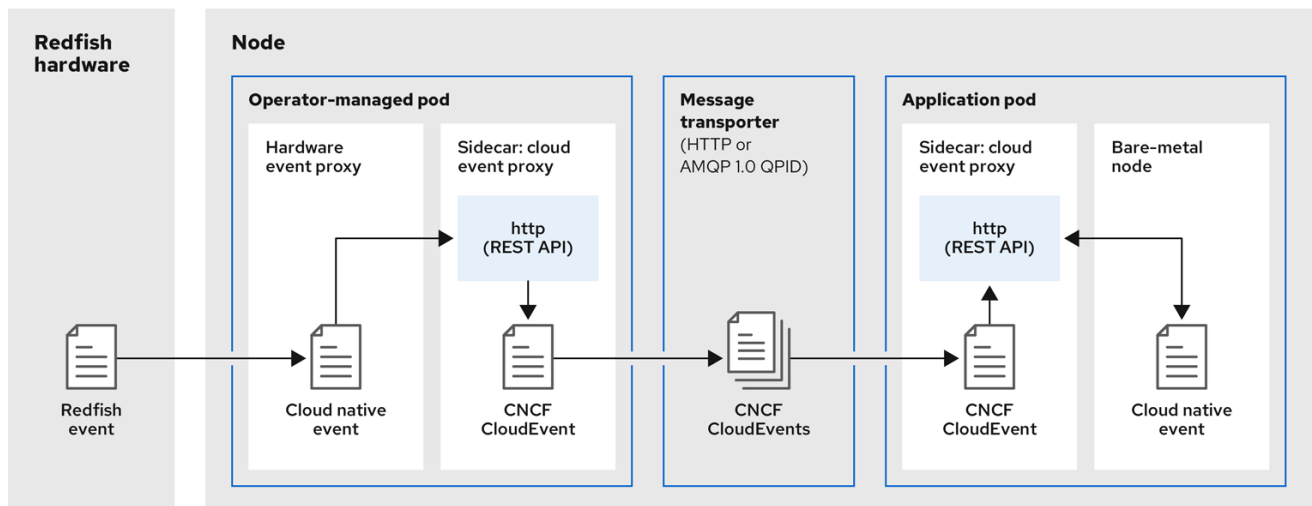
9.2.1.1. Operator-managed pod

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

9.2.1.2. Bare Metal Event Relay

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

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

9.2.1.3. Cloud native event

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

9.2.1.4. CNCF CloudEvents

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

9.2.1.5. HTTP transport or AMQP dispatch router

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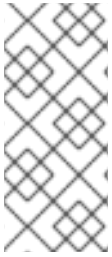
> **NOTE**
>
> HTTP transport is the default transport for PTP and bare-metal events. Use HTTP transport instead of AMQP for PTP and bare-metal events where possible. AMQ Interconnect is EOL from 30 June 2024. Extended life cycle support (ELS) for AMQ Interconnect ends 29 November 2029. For more information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

9.2.1.6. Cloud event proxy sidecar

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

9.2.2. Redfish message parsing service

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

9.2.3. Installing the Bare Metal Event Relay using the CLI

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

Prerequisites

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

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

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

Procedure

1. Create a namespace for the Bare Metal Event Relay.

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

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

   b. Create the **Namespace** CR:

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

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

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

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

   b. Create the **OperatorGroup** CR:

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

3. Subscribe to the Bare Metal Event Relay.

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

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

   b. Create the **Subscription** CR:

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

Verification

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

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

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

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

Prerequisites

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

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

Procedure

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

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

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

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

Verification

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

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

2. Ensure that **Bare Metal Event Relay**is listed in the project with a **Status** of **InstallSucceeded**.

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

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

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

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

9.3. INSTALLING THE AMQ MESSAGING BUS

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

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

Prerequisites

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

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

Procedure

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

Verification

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

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

Example output

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

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

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

Example output

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

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

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

9.4.1. Subscribing to bare-metal events

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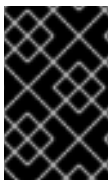
IMPORTANT

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

NOTE

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

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

Prerequisites

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

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

- Get the user name and password for the BMC.

- Deploy a bare-metal node with a Redfish-enabled Baseboard Management Controller (BMC) in your cluster, and enable Redfish events on the BMC.
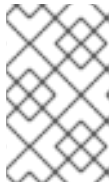
> **NOTE**
>
> Enabling Redfish events on specific hardware is outside the scope of this information. For more information about enabling Redfish events for your specific hardware, consult the BMC manufacturer documentation.

Procedure

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

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

   where:

   bmc_ip_address

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

   Example output

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

```
    "Status": {
      "Health": "OK",
      "HealthRollup": "OK",
      "State": "Enabled"
    },
    "Subscriptions": {
      "@odata.id": "/redfish/v1/EventService/Subscriptions"
    }
  }
}
```

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

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

Example output

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

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

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

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

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

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

   b. Create the **BMCEventSubscription** CR:

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

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

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

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

```
$ curl -i -k -X POST -H "Content-Type: application/json"  -d '{"Destination":
"https://<proxy_service_url>", "Protocol" : "Redfish", "EventTypes": ["Alert"], "Context":
"root"}' -u <bmc_username>:<password>
'https://<bmc_ip_address>/redfish/v1/EventService/Subscriptions' –v
```

where:

proxy_service_url

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

bmc_ip_address

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

Example output

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

9.4.2. Querying Redfish bare-metal event subscriptions with curl

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

Prerequisites

- Get the user name and password for the BMC.

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

Procedure

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

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

where:

bmc_ip_address

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

Example output

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

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

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

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

where:

bmc_ip_address

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

9.4.3. Creating the bare-metal event and Secret CRs

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

Prerequisites

- You have installed the OpenShift Container Platform CLI (**oc**).

- You have logged in as a user with **cluster-admin** privileges.

- You have installed the Bare Metal Event Relay.

- You have created a **BMCEventSubscription** CR for the BMC Redfish hardware.

Procedure

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

   > **NOTE**
   >
   > Multiple **HardwareEvent** resources are not permitted.

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

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

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

   > **NOTE**
   >
   > In OpenShift Container Platform 4.13 or later, you do not need to set the **spec.transportHost** field in the **HardwareEvent** resource when you use HTTP transport for bare-metal events. Set **transportHost** only when you use AMQP transport for bare-metal events.

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

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

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

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

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

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

   ```
   apiVersion: v1
   kind: Secret
   metadata:
     name: redfish-basic-auth
   type: Opaque
   stringData:  1
     username: <bmc_username>
   ```

```
password: <bmc_password>
# BMC host DNS or IP address
hostaddr: <bmc_host_ip_address>
```

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

b. Create the **Secret** CR:

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

Additional resources

- Persistent storage using local volumes

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

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

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

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

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

- **/api/ocloudNotifications/v1/subscriptions**

    - **POST**: Creates a new subscription

    - **GET**: Retrieves a list of subscriptions

- **/api/ocloudNotifications/v1/subscriptions/<subscription_id>**

    - **PUT**: Creates a new status ping request for the specified subscription ID

- **/api/ocloudNotifications/v1/health**

    - **GET**: Returns the health status of **ocloudNotifications** API

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

api/ocloudNotifications/v1/subscriptions
HTTP method
GET api/ocloudNotifications/v1/subscriptions

Description

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

Example API response

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

HTTP method
POST api/ocloudNotifications/v1/subscriptions

Description

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

Table 9.1. Query parameters

Parameter	Type
subscription	data

Example payload

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

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

Description

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

Table 9.2. Query parameters

Parameter	Type
<subscription_id>	string

Example API response

```
{
```

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

api/ocloudNotifications/v1/health/
HTTP method
GET api/ocloudNotifications/v1/health/

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

Example API response

```
OK
```

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

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

Prerequisites

- You have installed the OpenShift CLI (**oc**).

- You have logged in as a user with **cluster-admin** privileges.

- You have updated the PTP Operator or Bare Metal Event Relay to version 4.13+ which uses HTTP transport by default.

Procedure

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

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

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

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

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

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

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

10.1. WHAT HUGE PAGES DO

Memory is managed in blocks known as pages. On most systems, a page is 4Ki. 1Mi of memory is equal to 256 pages; 1Gi of memory is 256,000 pages, and so on. CPUs have a built-in memory management unit that manages a list of these pages in hardware. The Translation Lookaside Buffer (TLB) is a small hardware cache of virtual-to-physical page mappings. If the virtual address passed in a hardware instruction can be found in the TLB, the mapping can be determined quickly. If not, a TLB miss occurs, and the system falls back to slower, software-based address translation, resulting in performance issues. Since the size of the TLB is fixed, the only way to reduce the chance of a TLB miss is to increase the page size.

A huge page is a memory page that is larger than 4Ki. On x86_64 architectures, there are two common huge page sizes: 2Mi and 1Gi. Sizes vary on other architectures. To use huge pages, code must be written so that applications are aware of them. Transparent Huge Pages (THP) attempt to automate the management of huge pages without application knowledge, but they have limitations. In particular, they are limited to 2Mi page sizes. THP can lead to performance degradation on nodes with high memory utilization or fragmentation due to defragmenting efforts of THP, which can lock memory pages. For this reason, some applications may be designed to (or recommend) usage of pre-allocated huge pages instead of THP.

In OpenShift Container Platform, applications in a pod can allocate and consume pre-allocated huge pages.

10.2. HOW HUGE PAGES ARE CONSUMED BY APPS

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

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

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

```
      hugepages-2Mi: 100Mi ❶
      memory: "1Gi"
      cpu: "1"
 volumes:
 - name: hugepage
   emptyDir:
     medium: HugePages
```

❶ Specify the amount of memory for **hugepages** as the exact amount to be allocated. Do not specify this value as the amount of memory for **hugepages** multiplied by the size of the page. For example, given a huge page size of 2MB, if you want to use 100MB of huge-page-backed RAM for your application, then you would allocate 50 huge pages. OpenShift Container Platform handles the math for you. As in the above example, you can specify **100MB** directly.

Allocating huge pages of a specific size

Some platforms support multiple huge page sizes. To allocate huge pages of a specific size, precede the huge pages boot command parameters with a huge page size selection parameter **hugepagesz=<size>**. The **<size>** value must be specified in bytes with an optional scale suffix [**kKmMgG**]. The default huge page size can be defined with the **default_hugepagesz=<size>** boot parameter.

Huge page requirements

- Huge page requests must equal the limits. This is the default if limits are specified, but requests are not.

- Huge pages are isolated at a pod scope. Container isolation is planned in a future iteration.

- **EmptyDir** volumes backed by huge pages must not consume more huge page memory than the pod request.

- Applications that consume huge pages via **shmget()** with **SHM_HUGETLB** must run with a supplemental group that matches *proc/sys/vm/hugetlb_shm_group*.

10.3. CONSUMING HUGE PAGES RESOURCES USING THE DOWNWARD API

You can use the Downward API to inject information about the huge pages resources that are consumed by a container.

You can inject the resource allocation as environment variables, a volume plugin, or both. Applications that you develop and run in the container can determine the resources that are available by reading the environment variables or files in the specified volumes.

Procedure

1. Create a **hugepages-volume-pod.yaml** file that is similar to the following example:

```
apiVersion: v1
kind: Pod
metadata:
  generateName: hugepages-volume-
  labels:
    app: hugepages-example
```

```
spec:
  containers:
  - securityContext:
      capabilities:
        add: [ "IPC_LOCK" ]
    image: rhel7:latest
    command:
    - sleep
    - inf
    name: example
    volumeMounts:
    - mountPath: /dev/hugepages
      name: hugepage
    - mountPath: /etc/podinfo
      name: podinfo
    resources:
      limits:
        hugepages-1Gi: 2Gi
        memory: "1Gi"
        cpu: "1"
      requests:
        hugepages-1Gi: 2Gi
    env:
    - name: REQUESTS_HUGEPAGES_1GI <.>
      valueFrom:
        resourceFieldRef:
          containerName: example
          resource: requests.hugepages-1Gi
  volumes:
  - name: hugepage
    emptyDir:
      medium: HugePages
  - name: podinfo
    downwardAPI:
      items:
        - path: "hugepages_1G_request" <.>
          resourceFieldRef:
            containerName: example
            resource: requests.hugepages-1Gi
            divisor: 1Gi
```

<.> Specifies to read the resource use from **requests.hugepages-1Gi** and expose the value as the **REQUESTS_HUGEPAGES_1GI** environment variable. <.> Specifies to read the resource use from **requests.hugepages-1Gi** and expose the value as the file **/etc/podinfo/hugepages_1G_request**.

2. Create the pod from the **hugepages-volume-pod.yaml** file:

```
$ oc create -f hugepages-volume-pod.yaml
```

Verification

1. Check the value of the **REQUESTS_HUGEPAGES_1GI** environment variable:

```
$ oc exec -it $(oc get pods -l app=hugepages-example -o
jsonpath='{.items[0].metadata.name}') \
    -- env | grep REQUESTS_HUGEPAGES_1GI
```

Example output

```
REQUESTS_HUGEPAGES_1GI=2147483648
```

2. Check the value of the **/etc/podinfo/hugepages_1G_request** file:

```
$ oc exec -it $(oc get pods -l app=hugepages-example -o
jsonpath='{.items[0].metadata.name}') \
    -- cat /etc/podinfo/hugepages_1G_request
```

Example output

```
2
```

Additional resources

- [Allowing containers to consume Downward API objects](#)

10.4. CONFIGURING HUGE PAGES AT BOOT TIME

Nodes must pre-allocate huge pages used in an OpenShift Container Platform cluster. There are two ways of reserving huge pages: at boot time and at run time. Reserving at boot time increases the possibility of success because the memory has not yet been significantly fragmented. The Node Tuning Operator currently supports boot time allocation of huge pages on specific nodes.

Procedure

To minimize node reboots, the order of the steps below needs to be followed:

1. Label all nodes that need the same huge pages setting by a label.

```
$ oc label node <node_using_hugepages> node-role.kubernetes.io/worker-hp=
```

2. Create a file with the following content and name it **hugepages-tuned-boottime.yaml**:

```
apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
  name: hugepages ❶
  namespace: openshift-cluster-node-tuning-operator
spec:
  profile: ❷
  - data: |
      [main]
      summary=Boot time configuration for hugepages
      include=openshift-node
      [bootloader]
      cmdline_openshift_node_hugepages=hugepagesz=2M hugepages=50 ❸
```

```
    name: openshift-node-hugepages

    recommend:
    - machineConfigLabels: 4
        machineconfiguration.openshift.io/role: "worker-hp"
      priority: 30
      profile: openshift-node-hugepages
```

1 Set the **name** of the Tuned resource to **hugepages**.

2 Set the **profile** section to allocate huge pages.

3 Note the order of parameters is important as some platforms support huge pages of various sizes.

4 Enable machine config pool based matching.

3. Create the Tuned **hugepages** object

    ```
    $ oc create -f hugepages-tuned-boottime.yaml
    ```

4. Create a file with the following content and name it **hugepages-mcp.yaml**:

    ```
    apiVersion: machineconfiguration.openshift.io/v1
    kind: MachineConfigPool
    metadata:
      name: worker-hp
      labels:
        worker-hp: ""
    spec:
      machineConfigSelector:
        matchExpressions:
          - {key: machineconfiguration.openshift.io/role, operator: In, values: [worker,worker-hp]}
      nodeSelector:
        matchLabels:
          node-role.kubernetes.io/worker-hp: ""
    ```

5. Create the machine config pool:

    ```
    $ oc create -f hugepages-mcp.yaml
    ```

Given enough non-fragmented memory, all the nodes in the **worker-hp** machine config pool should now have 50 2Mi huge pages allocated.

```
$ oc get node <node_using_hugepages> -o jsonpath="{.status.allocatable.hugepages-2Mi}"
100Mi
```

> **NOTE**
>
> The TuneD bootloader plugin only supports Red Hat Enterprise Linux CoreOS (RHCOS) worker nodes.

10.5. DISABLING TRANSPARENT HUGE PAGES

Transparent Huge Pages (THP) attempt to automate most aspects of creating, managing, and using huge pages. Since THP automatically manages the huge pages, this is not always handled optimally for all types of workloads. THP can lead to performance regressions, since many applications handle huge pages on their own. Therefore, consider disabling THP. The following steps describe how to disable THP using the Node Tuning Operator (NTO).

Procedure

1. Create a file with the following content and name it **thp-disable-tuned.yaml**:

   ```
   apiVersion: tuned.openshift.io/v1
   kind: Tuned
   metadata:
     name: thp-workers-profile
     namespace: openshift-cluster-node-tuning-operator
   spec:
     profile:
     - data: |
         [main]
         summary=Custom tuned profile for OpenShift to turn off THP on worker nodes
         include=openshift-node

         [vm]
         transparent_hugepages=never
       name: openshift-thp-never-worker

     recommend:
     - match:
       - label: node-role.kubernetes.io/worker
       priority: 25
       profile: openshift-thp-never-worker
   ```

2. Create the Tuned object:

   ```
   $ oc create -f thp-disable-tuned.yaml
   ```

3. Check the list of active profiles:

   ```
   $ oc get profile -n openshift-cluster-node-tuning-operator
   ```

Verification

- Log in to one of the nodes and do a regular THP check to verify if the nodes applied the profile successfully:

  ```
  $ cat /sys/kernel/mm/transparent_hugepage/enabled
  ```

  Example output

  ```
  always madvise [never]
  ```

CHAPTER 11. LOW LATENCY TUNING

11.1. UNDERSTANDING LOW LATENCY

The emergence of Edge computing in the area of Telco / 5G plays a key role in reducing latency and congestion problems and improving application performance.

Simply put, latency determines how fast data (packets) moves from the sender to receiver and returns to the sender after processing by the receiver. Maintaining a network architecture with the lowest possible delay of latency speeds is key for meeting the network performance requirements of 5G. Compared to 4G technology, with an average latency of 50 ms, 5G is targeted to reach latency numbers of 1 ms or less. This reduction in latency boosts wireless throughput by a factor of 10.

Many of the deployed applications in the Telco space require low latency that can only tolerate zero packet loss. Tuning for zero packet loss helps mitigate the inherent issues that degrade network performance. For more information, see Tuning for Zero Packet Loss in Red Hat OpenStack Platform (RHOSP).

The Edge computing initiative also comes in to play for reducing latency rates. Think of it as being on the edge of the cloud and closer to the user. This greatly reduces the distance between the user and distant data centers, resulting in reduced application response times and performance latency.

Administrators must be able to manage their many Edge sites and local services in a centralized way so that all of the deployments can run at the lowest possible management cost. They also need an easy way to deploy and configure certain nodes of their cluster for real-time low latency and high-performance purposes. Low latency nodes are useful for applications such as Cloud-native Network Functions (CNF) and Data Plane Development Kit (DPDK).

OpenShift Container Platform currently provides mechanisms to tune software on an OpenShift Container Platform cluster for real-time running and low latency (around <20 microseconds reaction time). This includes tuning the kernel and OpenShift Container Platform set values, installing a kernel, and reconfiguring the machine. But this method requires setting up four different Operators and performing many configurations that, when done manually, is complex and could be prone to mistakes.

OpenShift Container Platform uses the Node Tuning Operator to implement automatic tuning to achieve low latency performance for OpenShift Container Platform applications. The cluster administrator uses this performance profile configuration that makes it easier to make these changes in a more reliable way. The administrator can specify whether to update the kernel to kernel-rt, reserve CPUs for cluster and operating system housekeeping duties, including pod infra containers, and isolate CPUs for application containers to run the workloads.

> **IMPORTANT**
>
> In OpenShift Container Platform 4.14, if you apply a performance profile to your cluster, all nodes in the cluster will reboot. This reboot includes control plane nodes and worker nodes that were not targeted by the performance profile. This is a known issue in OpenShift Container Platform 4.14 because this release uses Linux control group version 2 (cgroup v2) in alignment with RHEL 9. The low latency tuning features associated with the performance profile do not support cgroup v2, therefore the nodes reboot to switch back to the cgroup v1 configuration.
>
> To revert all nodes in the cluster to the cgroups v2 configuration, you must edit the **Node** resource. (OCPBUGS-16976)

> **NOTE**
>
> Currently, disabling CPU load balancing is not supported by cgroup v2. As a result, you might not get the desired behavior from performance profiles if you have cgroup v2 enabled. Enabling cgroup v2 is not recommended if you are using performance profiles.

OpenShift Container Platform also supports workload hints for the Node Tuning Operator that can tune the **PerformanceProfile** to meet the demands of different industry environments. Workload hints are available for **highPowerConsumption** (very low latency at the cost of increased power consumption) and **realTime** (priority given to optimum latency). A combination of **true/false** settings for these hints can be used to deal with application-specific workload profiles and requirements.

Workload hints simplify the fine-tuning of performance to industry sector settings. Instead of a "one size fits all" approach, workload hints can cater to usage patterns such as placing priority on:

- Low latency

- Real-time capability

- Efficient use of power

In an ideal world, all of those would be prioritized: in real life, some come at the expense of others. The Node Tuning Operator is now aware of the workload expectations and better able to meet the demands of the workload. The cluster admin can now specify into which use case that workload falls. The Node Tuning Operator uses the **PerformanceProfile** to fine tune the performance settings for the workload.

The environment in which an application is operating influences its behavior. For a typical data center with no strict latency requirements, only minimal default tuning is needed that enables CPU partitioning for some high performance workload pods. For data centers and workloads where latency is a higher priority, measures are still taken to optimize power consumption. The most complicated cases are clusters close to latency-sensitive equipment such as manufacturing machinery and software-defined radios. This last class of deployment is often referred to as Far edge. For Far edge deployments, ultra-low latency is the ultimate priority, and is achieved at the expense of power management.

11.1.1. About hyperthreading for low latency and real-time applications

Hyperthreading is an Intel processor technology that allows a physical CPU processor core to function as two logical cores, executing two independent threads simultaneously. Hyperthreading allows for better system throughput for certain workload types where parallel processing is beneficial. The default OpenShift Container Platform configuration expects hyperthreading to be enabled by default.

For telecommunications applications, it is important to design your application infrastructure to minimize latency as much as possible. Hyperthreading can slow performance times and negatively affect throughput for compute intensive workloads that require low latency. Disabling hyperthreading ensures predictable performance and can decrease processing times for these workloads.

> **NOTE**
>
> Hyperthreading implementation and configuration differs depending on the hardware you are running OpenShift Container Platform on. Consult the relevant host hardware tuning information for more details of the hyperthreading implementation specific to that hardware. Disabling hyperthreading can increase the cost per core of the cluster.

Additional resources

- [Configuring hyperthreading for a cluster](#)

11.2. PROVISIONING REAL-TIME AND LOW LATENCY WORKLOADS

Many industries and organizations need extremely high performance computing and might require low and predictable latency, especially in the financial and telecommunications industries. For these industries, with their unique requirements, OpenShift Container Platform provides the Node Tuning Operator to implement automatic tuning to achieve low latency performance and consistent response time for OpenShift Container Platform applications.

The cluster administrator can use this performance profile configuration to make these changes in a more reliable way. The administrator can specify whether to update the kernel to kernel-rt (real-time), reserve CPUs for cluster and operating system housekeeping duties, including pod infra containers, isolate CPUs for application containers to run the workloads, and disable unused CPUs to reduce power consumption.

> **WARNING**
>
> The usage of execution probes in conjunction with applications that require guaranteed CPUs can cause latency spikes. It is recommended to use other probes, such as a properly configured set of network probes, as an alternative.

> **NOTE**
>
> In earlier versions of OpenShift Container Platform, the Performance Addon Operator was used to implement automatic tuning to achieve low latency performance for OpenShift applications. In OpenShift Container Platform 4.11 and later, these functions are part of the Node Tuning Operator.

11.2.1. Known limitations for real-time

> **NOTE**
>
> In most deployments, kernel-rt is supported only on worker nodes when you use a standard cluster with three control plane nodes and three worker nodes. There are exceptions for compact and single nodes on OpenShift Container Platform deployments. For installations on a single node, kernel-rt is supported on the single control plane node.

To fully utilize the real-time mode, the containers must run with elevated privileges. See [Set capabilities for a Container](#) for information on granting privileges.

OpenShift Container Platform restricts the allowed capabilities, so you might need to create a **SecurityContext** as well.

> **NOTE**
>
> This procedure is fully supported with bare metal installations using Red Hat Enterprise Linux CoreOS (RHCOS) systems.

Establishing the right performance expectations refers to the fact that the real-time kernel is not a panacea. Its objective is consistent, low-latency determinism offering predictable response times. There is some additional kernel overhead associated with the real-time kernel. This is due primarily to handling hardware interruptions in separately scheduled threads. The increased overhead in some workloads results in some degradation in overall throughput. The exact amount of degradation is very workload dependent, ranging from 0% to 30%. However, it is the cost of determinism.

11.2.2. Provisioning a worker with real-time capabilities

1. Optional: Add a node to the OpenShift Container Platform cluster. See Setting BIOS parameters for system tuning.

2. Add the label **worker-rt** to the worker nodes that require the real-time capability by using the **oc** command.

3. Create a new machine config pool for real-time nodes:

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  name: worker-rt
  labels:
    machineconfiguration.openshift.io/role: worker-rt
spec:
  machineConfigSelector:
    matchExpressions:
      - {
          key: machineconfiguration.openshift.io/role,
          operator: In,
          values: [worker, worker-rt],
        }
  paused: false
  nodeSelector:
    matchLabels:
      node-role.kubernetes.io/worker-rt: ""
```

Note that a machine config pool worker-rt is created for group of nodes that have the label **worker-rt**.

4. Add the node to the proper machine config pool by using node role labels.
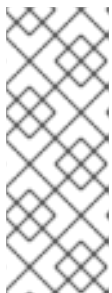
> **NOTE**
>
> You must decide which nodes are configured with real-time workloads. You could configure all of the nodes in the cluster, or a subset of the nodes. The Node Tuning Operator that expects all of the nodes are part of a dedicated machine config pool. If you use all of the nodes, you must point the Node Tuning Operator to the worker node role label. If you use a subset, you must group the nodes into a new machine config pool.

5. Create the **PerformanceProfile** with the proper set of housekeeping cores and **realTimeKernel: enabled: true**.

6. You must set **machineConfigPoolSelector** in **PerformanceProfile**:

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
 name: example-performanceprofile
spec:
...
  realTimeKernel:
    enabled: true
  nodeSelector:
    node-role.kubernetes.io/worker-rt: ""
  machineConfigPoolSelector:
    machineconfiguration.openshift.io/role: worker-rt
```

7. Verify that a matching machine config pool exists with a label:

```
$ oc describe mcp/worker-rt
```

Example output

```
Name:        worker-rt
Namespace:
Labels:      machineconfiguration.openshift.io/role=worker-rt
```

8. OpenShift Container Platform will start configuring the nodes, which might involve multiple reboots. Wait for the nodes to settle. This can take a long time depending on the specific hardware you use, but 20 minutes per node is expected.

9. Verify everything is working as expected.

11.2.3. Verifying the real-time kernel installation

Use this command to verify that the real-time kernel is installed:

```
$ oc get node -o wide
```

Note the worker with the role **worker-rt** that contains the string **4.18.0-305.30.1.rt7.102.el8_4.x86_64 cri-o://1.28.5-99.rhaos4.10.gitc3131de.el8**:

```
NAME                     STATUS  ROLES          AGE VERSION              INTERNAL-IP
EXTERNAL-IP   OS-IMAGE                          KERNEL-VERSION
CONTAINER-RUNTIME
rt-worker-0.example.com         Ready  worker,worker-rt  5d17h  v1.28.5
128.66.135.107  <none>        Red Hat Enterprise Linux CoreOS 46.82.202008252340-0 (Ootpa)
4.18.0-305.30.1.rt7.102.el8_4.x86_64   cri-o://1.28.5-99.rhaos4.10.gitc3131de.el8
[...]
```

11.2.4. Creating a workload that works in real-time

Use the following procedures for preparing a workload that will use real-time capabilities.

Procedure

1. Create a pod with a QoS class of **Guaranteed**.

2. Optional: Disable CPU load balancing for DPDK.

3. Assign a proper node selector.

When writing your applications, follow the general recommendations described in Application tuning and deployment.

11.2.5. Creating a pod with a QoS class of **Guaranteed**

Keep the following in mind when you create a pod that is given a QoS class of **Guaranteed**:

- Every container in the pod must have a memory limit and a memory request, and they must be the same.

- Every container in the pod must have a CPU limit and a CPU request, and they must be the same.

The following example shows the configuration file for a pod that has one container. The container has a memory limit and a memory request, both equal to 200 MiB. The container has a CPU limit and a CPU request, both equal to 1 CPU.

```
apiVersion: v1
kind: Pod
metadata:
  name: qos-demo
  namespace: qos-example
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
  - name: qos-demo-ctr
    image: <image-pull-spec>
    resources:
      limits:
        memory: "200Mi"
        cpu: "1"
      requests:
        memory: "200Mi"
        cpu: "1"
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
```

1. Create the pod:

   ```
   $ oc  apply -f qos-pod.yaml --namespace=qos-example
   ```

2. View detailed information about the pod:

   ```
   $ oc get pod qos-demo --namespace=qos-example --output=yaml
   ```

   Example output

```
spec:
  containers:
    ...
status:
  qosClass: Guaranteed
```

> **NOTE**
>
> If a container specifies its own memory limit, but does not specify a memory request, OpenShift Container Platform automatically assigns a memory request that matches the limit. Similarly, if a container specifies its own CPU limit, but does not specify a CPU request, OpenShift Container Platform automatically assigns a CPU request that matches the limit.

11.2.6. Optional: Disabling CPU load balancing for DPDK

Functionality to disable or enable CPU load balancing is implemented on the CRI-O level. The code under the CRI-O disables or enables CPU load balancing only when the following requirements are met.

- The pod must use the **performance-<profile-name>** runtime class. You can get the proper name by looking at the status of the performance profile, as shown here:

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
...
status:
  ...
  runtimeClass: performance-manual
```

> **NOTE**
>
> Currently, disabling CPU load balancing is not supported with cgroup v2.

The Node Tuning Operator is responsible for the creation of the high-performance runtime handler config snippet under relevant nodes and for creation of the high-performance runtime class under the cluster. It will have the same content as default runtime handler except it enables the CPU load balancing configuration functionality.

To disable the CPU load balancing for the pod, the **Pod** specification must include the following fields:

```
apiVersion: v1
kind: Pod
metadata:
  ...
  annotations:
    ...
    cpu-load-balancing.crio.io: "disable"
    ...
  ...
spec:
  ...
  runtimeClassName: performance-<profile_name>
  ...
```

> **NOTE**
>
> Only disable CPU load balancing when the CPU manager static policy is enabled and for pods with guaranteed QoS that use whole CPUs. Otherwise, disabling CPU load balancing can affect the performance of other containers in the cluster.

11.2.7. Assigning a proper node selector

The preferred way to assign a pod to nodes is to use the same node selector the performance profile used, as shown here:

```
apiVersion: v1
kind: Pod
metadata:
  name: example
spec:
  # ...
  nodeSelector:
    node-role.kubernetes.io/worker-rt: ""
```

For more information, see Placing pods on specific nodes using node selectors .

11.2.8. Scheduling a workload onto a worker with real-time capabilities

Use label selectors that match the nodes attached to the machine config pool that was configured for low latency by the Node Tuning Operator. For more information, see Assigning pods to nodes .

11.2.9. Reducing power consumption by taking CPUs offline

You can generally anticipate telecommunication workloads. When not all of the CPU resources are required, the Node Tuning Operator allows you take unused CPUs offline to reduce power consumption by manually updating the performance profile.

To take unused CPUs offline, you must perform the following tasks:

1. Set the offline CPUs in the performance profile and save the contents of the YAML file:

   Example performance profile with offlined CPUs

   ```
   apiVersion: performance.openshift.io/v2
   kind: PerformanceProfile
   metadata:
     name: performance
   spec:
     additionalKernelArgs:
     - nmi_watchdog=0
     - audit=0
     - mce=off
     - processor.max_cstate=1
     - intel_idle.max_cstate=0
     - idle=poll
     cpu:
       isolated: "2-23,26-47"
       reserved: "0,1,24,25"
   ```

```
    offlined: "48-59" 1
  nodeSelector:
    node-role.kubernetes.io/worker-cnf: ""
  numa:
    topologyPolicy: single-numa-node
  realTimeKernel:
    enabled: true
```

1   Optional. You can list CPUs in the **offlined** field to take the specified CPUs offline.

2. Apply the updated profile by running the following command:

```
$ oc apply -f my-performance-profile.yaml
```

11.2.10. Optional: Power saving configurations

You can enable power savings for a node that has low priority workloads that are colocated with high priority workloads without impacting the latency or throughput of the high priority workloads. Power saving is possible without modifications to the workloads themselves.
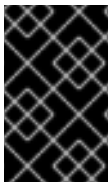
> **IMPORTANT**
>
> The feature is supported on Intel Ice Lake and later generations of Intel CPUs. The capabilities of the processor might impact the latency and throughput of the high priority workloads.

When you configure a node with a power saving configuration, you must configure high priority workloads with performance configuration at the pod level, which means that the configuration applies to all the cores used by the pod.

By disabling P-states and C-states at the pod level, you can configure high priority workloads for best performance and lowest latency.

Table 11.1. Configuration for high priority workloads

Annotation	Possible Values	Description
cpu-c-states.crio.io:	**"enable"** **"disable"** **"max_latency:microseconds"**	This annotation allows you to enable or disable C-states for each CPU. Alternatively, you can also specify a maximum latency in microseconds for the C-states. For example, enable C-states with a maximum latency of 10 microseconds with the setting **cpu-c-states.crio.io**: **"max_latency:10"**. Set the value to **"disable"** to provide the best performance for a pod.
cpu-freq-governor.crio.io:	Any supported **cpufreq governor**.	Sets the **cpufreq** governor for each CPU. The **"performance"** governor is recommended for high priority workloads.

Prerequisites

- You enabled C-states and OS-controlled P-states in the BIOS

Procedure

1. Generate a **PerformanceProfile** with **per-pod-power-management** set to **true**:

```
$ podman run --entrypoint performance-profile-creator -v \
/must-gather:/must-gather:z registry.redhat.io/openshift4/ose-cluster-node-tuning-
operator:v4.15 \
--mcp-name=worker-cnf --reserved-cpu-count=20 --rt-kernel=true \
--split-reserved-cpus-across-numa=false --topology-manager-policy=single-numa-node \
--must-gather-dir-path /must-gather -power-consumption-mode=low-latency \ 1
--per-pod-power-management=true > my-performance-profile.yaml
```

1 The **power-consumption-mode** must be **default** or **low-latency** when the **per-pod-power-management** is set to **true**.

Example PerformanceProfile with perPodPowerManagement

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
    name: performance
spec:
    [.....]
    workloadHints:
      realTime: true
      highPowerConsumption: false
      perPodPowerManagement: true
```

2. Set the default **cpufreq** governor as an additional kernel argument in the **PerformanceProfile** custom resource (CR):

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
    name: performance
spec:
    ...
    additionalKernelArgs:
    - cpufreq.default_governor=schedutil 1
```

1 Using the **schedutil** governor is recommended, however, you can use other governors such as the **ondemand** or **powersave** governors.

3. Set the maximum CPU frequency in the **TunedPerformancePatch** CR:

```
spec:
  profile:
  - data: |
      [sysfs]
      /sys/devices/system/cpu/intel_pstate/max_perf_pct = <x> 1
```

1   The **max_perf_pct** controls the maximum frequency the **cpufreq** driver is allowed to set as a percentage of the maximum supported cpu frequency. This value applies to all CPUs. You can check the maximum supported frequency in **/sys/devices/system/cpu/cpu0/cpufreq/cpuinfo_max_freq**. As a starting point, you can use a percentage that caps all CPUs at the **All Cores Turbo** frequency. The **All Cores Turbo** frequency is the frequency that all cores will run at when the cores are all fully occupied.

4. Add the desired annotations to your high priority workload pods. The annotations override the **default** settings.

   Example high priority workload annotation

   ```
   apiVersion: v1
   kind: Pod
   metadata:
     ...
     annotations:
       ...
       cpu-c-states.crio.io: "disable"
       cpu-freq-governor.crio.io: "performance"
       ...
     ...
   spec:
     ...
     runtimeClassName: performance-<profile_name>
     ...
   ```

5. Restart the pods.

Additional resources

- For more information about recommended firmware configuration, see Recommended firmware configuration for vDU cluster hosts.

11.2.11. Managing device interrupt processing for guaranteed pod isolated CPUs

The Node Tuning Operator can manage host CPUs by dividing them into reserved CPUs for cluster and operating system housekeeping duties, including pod infra containers, and isolated CPUs for application containers to run the workloads. This allows you to set CPUs for low latency workloads as isolated.

Device interrupts are load balanced between all isolated and reserved CPUs to avoid CPUs being overloaded, with the exception of CPUs where there is a guaranteed pod running. Guaranteed pod CPUs are prevented from processing device interrupts when the relevant annotations are set for the pod.

In the performance profile, **globallyDisableIrqLoadBalancing** is used to manage whether device interrupts are processed or not. For certain workloads, the reserved CPUs are not always sufficient for dealing with device interrupts, and for this reason, device interrupts are not globally disabled on the isolated CPUs. By default, Node Tuning Operator does not disable device interrupts on isolated CPUs.

To achieve low latency for workloads, some (but not all) pods require the CPUs they are running on to not process device interrupts. A pod annotation, **irq-load-balancing.crio.io**, is used to define whether device interrupts are processed or not. When configured, CRI-O disables device interrupts only as long

as the pod is running.

11.2.11.1. Disabling CPU CFS quota

To reduce CPU throttling for individual guaranteed pods, create a pod specification with the annotation **cpu-quota.crio.io: "disable"**. This annotation disables the CPU completely fair scheduler (CFS) quota at the pod run time. The following pod specification contains this annotation:

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    cpu-quota.crio.io: "disable"
spec:
  runtimeClassName: performance-<profile_name>
...
```

> **NOTE**
>
> Only disable CPU CFS quota when the CPU manager static policy is enabled and for pods with guaranteed QoS that use whole CPUs. Otherwise, disabling CPU CFS quota can affect the performance of other containers in the cluster.

11.2.11.2. Disabling global device interrupts handling in Node Tuning Operator

To configure Node Tuning Operator to disable global device interrupts for the isolated CPU set, set the **globallyDisableIrqLoadBalancing** field in the performance profile to **true**. When **true**, conflicting pod annotations are ignored. When **false**, IRQ loads are balanced across all CPUs.

A performance profile snippet illustrates this setting:

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: manual
spec:
  globallyDisableIrqLoadBalancing: true
...
```

11.2.11.3. Disabling interrupt processing for individual pods

To disable interrupt processing for individual pods, ensure that **globallyDisableIrqLoadBalancing** is set to **false** in the performance profile. Then, in the pod specification, set the **irq-load-balancing.crio.io** pod annotation to **disable**. The following pod specification contains this annotation:

```
apiVersion: performance.openshift.io/v2
kind: Pod
metadata:
  annotations:
    irq-load-balancing.crio.io: "disable"
spec:
  runtimeClassName: performance-<profile_name>
...
```

11.2.12. Upgrading the performance profile to use device interrupt processing

When you upgrade the Node Tuning Operator performance profile custom resource definition (CRD) from v1 or v1alpha1 to v2, **globallyDisableIrqLoadBalancing** is set to **true** on existing profiles.

> **NOTE**
>
> **globallyDisableIrqLoadBalancing** toggles whether IRQ load balancing will be disabled for the Isolated CPU set. When the option is set to **true** it disables IRQ load balancing for the Isolated CPU set. Setting the option to **false** allows the IRQs to be balanced across all CPUs.

11.2.12.1. Supported API Versions

The Node Tuning Operator supports **v2**, **v1**, and **v1alpha1** for the performance profile **apiVersion** field. The v1 and v1alpha1 APIs are identical. The v2 API includes an optional boolean field **globallyDisableIrqLoadBalancing** with a default value of **false**.

11.2.12.1.1. Upgrading Node Tuning Operator API from v1alpha1 to v1

When upgrading Node Tuning Operator API version from v1alpha1 to v1, the v1alpha1 performance profiles are converted on-the-fly using a "None" Conversion strategy and served to the Node Tuning Operator with API version v1.

11.2.12.1.2. Upgrading Node Tuning Operator API from v1alpha1 or v1 to v2

When upgrading from an older Node Tuning Operator API version, the existing v1 and v1alpha1 performance profiles are converted using a conversion webhook that injects the **globallyDisableIrqLoadBalancing** field with a value of **true**.

11.3. TUNING NODES FOR LOW LATENCY WITH THE PERFORMANCE PROFILE

The performance profile lets you control latency tuning aspects of nodes that belong to a certain machine config pool. After you specify your settings, the **PerformanceProfile** object is compiled into multiple objects that perform the actual node level tuning:

- A **MachineConfig** file that manipulates the nodes.

- A **KubeletConfig** file that configures the Topology Manager, the CPU Manager, and the OpenShift Container Platform nodes.

- The Tuned profile that configures the Node Tuning Operator.

You can use a performance profile to specify whether to update the kernel to kernel-rt, to allocate huge pages, and to partition the CPUs for performing housekeeping duties or running workloads.
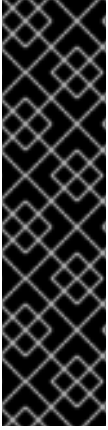
IMPORTANT

In OpenShift Container Platform 4.15, if you apply a performance profile to your cluster, all nodes in the cluster will reboot. This reboot includes control plane nodes and worker nodes that were not targeted by the performance profile. This is a known issue in OpenShift Container Platform 4.15 because this release uses Linux control group version 2 (cgroup v2) in alignment with RHEL 9. The low latency tuning features associated with the performance profile do not support cgroup v2, therefore the nodes reboot to switch back to the cgroup v1 configuration.

To revert all nodes in the cluster to the cgroups v2 configuration, you must edit the **Node** resource. (**OCPBUGS-16976**)

NOTE

You can manually create the **PerformanceProfile** object or use the Performance Profile Creator (PPC) to generate a performance profile. See the additional resources below for more information on the PPC.

Sample performance profile

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
 name: performance
spec:
 cpu:
  isolated: "4-15"  1
  reserved: "0-3"   2
 hugepages:
  defaultHugepagesSize: "1G"
  pages:
  - size: "1G"
    count: 16
    node: 0
 realTimeKernel:
  enabled: true    3
 numa:             4
  topologyPolicy: "best-effort"
 nodeSelector:
  node-role.kubernetes.io/worker-cnf: ""  5
```

1. Use this field to isolate specific CPUs to use with application containers for workloads. Set an even number of isolated CPUs to enable the pods to run without errors when hyperthreading is enabled.

2. Use this field to reserve specific CPUs to use with infra containers for housekeeping.

3. Use this field to install the real-time kernel on the node. Valid values are **true** or **false**. Setting the **true** value installs the real-time kernel.

4. Use this field to configure the topology manager policy. Valid values are **none** (default), **best-effort**, **restricted**, and **single-numa-node**. For more information, see Topology Manager Policies.

5. Use this field to specify a node selector to apply the performance profile to specific nodes.

Additional resources

- For information on using the Performance Profile Creator (PPC) to generate a performance profile, see Creating a performance profile .

11.3.1. Configuring huge pages

Nodes must pre-allocate huge pages used in an OpenShift Container Platform cluster. Use the Node Tuning Operator to allocate huge pages on a specific node.

OpenShift Container Platform provides a method for creating and allocating huge pages. Node Tuning Operator provides an easier method for doing this using the performance profile.

For example, in the **hugepages pages** section of the performance profile, you can specify multiple blocks of **size**, **count**, and, optionally, **node**:

```
hugepages:
   defaultHugepagesSize: "1G"
   pages:
   - size: "1G"
     count: 4
     node: 0
```
 1

1 **node** is the NUMA node in which the huge pages are allocated. If you omit   **node**, the pages are evenly spread across all NUMA nodes.

> **NOTE**
>
> Wait for the relevant machine config pool status that indicates the update is finished.

These are the only configuration steps you need to do to allocate huge pages.

Verification

- To verify the configuration, see the **/proc/meminfo** file on the node:

  ```
  $ oc debug node/ip-10-0-141-105.ec2.internal
  ```

  ```
  # grep -i huge /proc/meminfo
  ```

  Example output

  ```
  AnonHugePages:    ###### ##
  ShmemHugePages:        0 kB
  HugePages_Total:      2
  HugePages_Free:       2
  HugePages_Rsvd:       0
  HugePages_Surp:       0
  Hugepagesize:      #### ##
  Hugetlb:           #### ##
  ```

- Use **oc describe** to report the new size:

```
$ oc describe node worker-0.ocp4poc.example.com | grep -i huge
```

Example output

```
                    hugepages-1g=true
hugepages-###:  ###
hugepages-###:  ###
```

11.3.2. Allocating multiple huge page sizes

You can request huge pages with different sizes under the same container. This allows you to define more complicated pods consisting of containers with different huge page size needs.

For example, you can define sizes **1G** and **2M** and the Node Tuning Operator will configure both sizes on the node, as shown here:

```
spec:
  hugepages:
    defaultHugepagesSize: 1G
    pages:
    - count: 1024
      node: 0
      size: 2M
    - count: 4
      node: 1
      size: 1G
```

11.3.3. Configuring a node for IRQ dynamic load balancing

Configure a cluster node for IRQ dynamic load balancing to control which cores can receive device interrupt requests (IRQ).

Prerequisites

- For core isolation, all server hardware components must support IRQ affinity. To check if the hardware components of your server support IRQ affinity, view the server's hardware specifications or contact your hardware provider.

Procedure

1. Log in to the OpenShift Container Platform cluster as a user with cluster-admin privileges.

2. Set the performance profile **apiVersion** to use **performance.openshift.io/v2**.

3. Remove the **globallyDisableIrqLoadBalancing** field or set it to **false**.

4. Set the appropriate isolated and reserved CPUs. The following snippet illustrates a profile that reserves 2 CPUs. IRQ load-balancing is enabled for pods running on the **isolated** CPU set:

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: dynamic-irq-profile
```

```
spec:
  cpu:
    isolated: 2-5
    reserved: 0-1
...
```
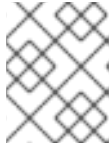
> **NOTE**
>
> When you configure reserved and isolated CPUs, the infra containers in pods use the reserved CPUs and the application containers use the isolated CPUs.

5. Create the pod that uses exclusive CPUs, and set **irq-load-balancing.crio.io** and **cpu-quota.crio.io** annotations to **disable**. For example:

```
apiVersion: v1
kind: Pod
metadata:
  name: dynamic-irq-pod
  annotations:
     irq-load-balancing.crio.io: "disable"
     cpu-quota.crio.io: "disable"
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
  - name: dynamic-irq-pod
    image: "registry.redhat.io/openshift4/cnf-tests-rhel8:v4.15"
    command: ["sleep", "10h"]
    resources:
      requests:
        cpu: 2
        memory: "200M"
      limits:
        cpu: 2
        memory: "200M"
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
  nodeSelector:
    node-role.kubernetes.io/worker-cnf: ""
  runtimeClassName: performance-dynamic-irq-profile
# ...
```

6. Enter the pod **runtimeClassName** in the form performance-<profile_name>, where <profile_name> is the **name** from the **PerformanceProfile** YAML, in this example, **performance-dynamic-irq-profile**.

7. Set the node selector to target a cnf-worker.

8. Ensure the pod is running correctly. Status should be **running**, and the correct cnf-worker node should be set:

```
$ oc get pod -o wide
```

Expected output

```
NAME            READY  STATUS   RESTARTS  AGE    IP          NODE
NOMINATED NODE   READINESS GATES
dynamic-irq-pod  1/1    Running  0         5h33m  <ip-address>  <node-name>  <none>
<none>
```

9. Get the CPUs that the pod configured for IRQ dynamic load balancing runs on:

```
$ oc exec -it dynamic-irq-pod -- /bin/bash -c "grep Cpus_allowed_list /proc/self/status | awk
'{print $2}'"
```

Expected output

```
Cpus_allowed_list:  2-3
```

10. Ensure the node configuration is applied correctly. Log in to the node to verify the configuration.

```
$ oc debug node/<node-name>
```

Expected output

```
Starting pod/<node-name>-debug ...
To use host binaries, run `chroot /host`

Pod IP: <ip-address>
If you don't see a command prompt, try pressing enter.

sh-4.4#
```

11. Verify that you can use the node file system:

```
sh-4.4# chroot /host
```

Expected output

```
sh-4.4#
```

12. Ensure the default system CPU affinity mask does not include the **dynamic-irq-pod** CPUs, for example, CPUs 2 and 3.

```
$ cat /proc/irq/default_smp_affinity
```

Example output

```
33
```

13. Ensure the system IRQs are not configured to run on the **dynamic-irq-pod** CPUs:

```
find /proc/irq/ -name smp_affinity_list -exec sh -c 'i="$1"; mask=$(cat $i); file=$(echo $i); echo
$file: $mask' _ {} \;
```

Example output

```
/proc/irq/0/smp_affinity_list: 0-5
/proc/irq/1/smp_affinity_list: 5
/proc/irq/2/smp_affinity_list: 0-5
/proc/irq/3/smp_affinity_list: 0-5
/proc/irq/4/smp_affinity_list: 0
/proc/irq/5/smp_affinity_list: 0-5
/proc/irq/6/smp_affinity_list: 0-5
/proc/irq/7/smp_affinity_list: 0-5
/proc/irq/8/smp_affinity_list: 4
/proc/irq/9/smp_affinity_list: 4
/proc/irq/10/smp_affinity_list: 0-5
/proc/irq/11/smp_affinity_list: 0
/proc/irq/12/smp_affinity_list: 1
/proc/irq/13/smp_affinity_list: 0-5
/proc/irq/14/smp_affinity_list: 1
/proc/irq/15/smp_affinity_list: 0
/proc/irq/24/smp_affinity_list: 1
/proc/irq/25/smp_affinity_list: 1
/proc/irq/26/smp_affinity_list: 1
/proc/irq/27/smp_affinity_list: 5
/proc/irq/28/smp_affinity_list: 1
/proc/irq/29/smp_affinity_list: 0
/proc/irq/30/smp_affinity_list: 0-5
```

11.3.4. About support of IRQ affinity setting

Some IRQ controllers lack support for IRQ affinity setting and will always expose all online CPUs as the IRQ mask. These IRQ controllers effectively run on CPU 0.

The following are examples of drivers and hardware that Red Hat are aware lack support for IRQ affinity setting. The list is, by no means, exhaustive:

- Some RAID controller drivers, such as **megaraid_sas**

- Many non-volatile memory express (NVMe) drivers

- Some LAN on motherboard (LOM) network controllers

- The driver uses **managed_irqs**

> **NOTE**
>
> The reason they do not support IRQ affinity setting might be associated with factors such as the type of processor, the IRQ controller, or the circuitry connections in the motherboard.

If the effective affinity of any IRQ is set to an isolated CPU, it might be a sign of some hardware or driver not supporting IRQ affinity setting. To find the effective affinity, log in to the host and run the following command:

```
$ find /proc/irq -name effective_affinity -printf "%p: " -exec cat {} \;
```

Example output

```
/proc/irq/0/effective_affinity: 1
/proc/irq/1/effective_affinity: 8
/proc/irq/2/effective_affinity: 0
/proc/irq/3/effective_affinity: 1
/proc/irq/4/effective_affinity: 2
/proc/irq/5/effective_affinity: 1
/proc/irq/6/effective_affinity: 1
/proc/irq/7/effective_affinity: 1
/proc/irq/8/effective_affinity: 1
/proc/irq/9/effective_affinity: 2
/proc/irq/10/effective_affinity: 1
/proc/irq/11/effective_affinity: 1
/proc/irq/12/effective_affinity: 4
/proc/irq/13/effective_affinity: 1
/proc/irq/14/effective_affinity: 1
/proc/irq/15/effective_affinity: 1
/proc/irq/24/effective_affinity: 2
/proc/irq/25/effective_affinity: 4
/proc/irq/26/effective_affinity: 2
/proc/irq/27/effective_affinity: 1
/proc/irq/28/effective_affinity: 8
/proc/irq/29/effective_affinity: 4
/proc/irq/30/effective_affinity: 4
/proc/irq/31/effective_affinity: 8
/proc/irq/32/effective_affinity: 8
/proc/irq/33/effective_affinity: 1
/proc/irq/34/effective_affinity: 2
```

Some drivers use **managed_irqs**, whose affinity is managed internally by the kernel and userspace cannot change the affinity. In some cases, these IRQs might be assigned to isolated CPUs. For more information about **managed_irqs**, see Affinity of managed interrupts cannot be changed even if they target isolated CPU.

11.3.5. Configuring hyperthreading for a cluster

To configure hyperthreading for an OpenShift Container Platform cluster, set the CPU threads in the performance profile to the same cores that are configured for the reserved or isolated CPU pools.

> **NOTE**
>
> If you configure a performance profile, and subsequently change the hyperthreading configuration for the host, ensure that you update the CPU **isolated** and **reserved** fields in the **PerformanceProfile** YAML to match the new configuration.

> **WARNING**
>
> Disabling a previously enabled host hyperthreading configuration can cause the CPU core IDs listed in the **PerformanceProfile** YAML to be incorrect. This incorrect configuration can cause the node to become unavailable because the listed CPUs can no longer be found.

Prerequisites

- Access to the cluster as a user with the **cluster-admin** role.

- Install the OpenShift CLI (oc).

Procedure

1. Ascertain which threads are running on what CPUs for the host you want to configure. You can view which threads are running on the host CPUs by logging in to the cluster and running the following command:

   ```
   $ lscpu --all --extended
   ```

   Example output

   ```
   CPU NODE SOCKET CORE L1d:L1i:L2:L3 ONLINE MAXMHZ    MINMHZ
   0   0    0      0    0:0:0:0       yes    4800.0000 400.0000
   1   0    0      1    1:1:1:0       yes    4800.0000 400.0000
   2   0    0      2    2:2:2:0       yes    4800.0000 400.0000
   3   0    0      3    3:3:3:0       yes    4800.0000 400.0000
   4   0    0      0    0:0:0:0       yes    4800.0000 400.0000
   5   0    0      1    1:1:1:0       yes    4800.0000 400.0000
   6   0    0      2    2:2:2:0       yes    4800.0000 400.0000
   7   0    0      3    3:3:3:0       yes    4800.0000 400.0000
   ```

   In this example, there are eight logical CPU cores running on four physical CPU cores. CPU0 and CPU4 are running on physical Core0, CPU1 and CPU5 are running on physical Core 1, and so on.

   Alternatively, to view the threads that are set for a particular physical CPU core (**cpu0** in the example below), open a command prompt and run the following:

   ```
   $ cat /sys/devices/system/cpu/cpu0/topology/thread_siblings_list
   ```

   Example output

   ```
   0-4
   ```

2. Apply the isolated and reserved CPUs in the **PerformanceProfile** YAML. For example, you can set logical cores CPU0 and CPU4 as **isolated**, and logical cores CPU1 to CPU3 and CPU5 to CPU7 as **reserved**. When you configure reserved and isolated CPUs, the infra containers in

pods use the reserved CPUs and the application containers use the isolated CPUs.

```
...
  cpu:
    isolated: 0,4
    reserved: 1-3,5-7
...
```
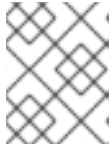
> **NOTE**
>
> The reserved and isolated CPU pools must not overlap and together must span all available cores in the worker node.

> **IMPORTANT**
>
> Hyperthreading is enabled by default on most Intel processors. If you enable hyperthreading, all threads processed by a particular core must be isolated or processed on the same core.

11.3.5.1. Disabling hyperthreading for low latency applications

When configuring clusters for low latency processing, consider whether you want to disable hyperthreading before you deploy the cluster. To disable hyperthreading, do the following:

1. Create a performance profile that is appropriate for your hardware and topology.

2. Set **nosmt** as an additional kernel argument. The following example performance profile illustrates this setting:

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: example-performanceprofile
spec:
  additionalKernelArgs:
    - nmi_watchdog=0
    - audit=0
    - mce=off
    - processor.max_cstate=1
    - idle=poll
    - intel_idle.max_cstate=0
    - nosmt
  cpu:
    isolated: 2-3
    reserved: 0-1
  hugepages:
    defaultHugepagesSize: 1G
    pages:
      - count: 2
        node: 0
        size: 1G
  nodeSelector:
    node-role.kubernetes.io/performance: "
  realTimeKernel:
    enabled: true
```

> **NOTE**
>
> When you configure reserved and isolated CPUs, the infra containers in pods use the reserved CPUs and the application containers use the isolated CPUs.

11.3.6. Understanding workload hints

The following table describes how combinations of power consumption and real-time settings impact on latency.
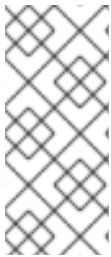
> **NOTE**
>
> The following workload hints can be configured manually. You can also work with workload hints using the Performance Profile Creator. For more information about the performance profile, see the "Creating a performance profile" section. If the workload hint is configured manually and the **realTime** workload hint is not explicitly set then it defaults to **true**.

Performance Profile creator setting	Hint	Environment	Description
Default	workloadHints: highPowerConsumption: false realTime: false	High throughput cluster without latency requirements	Performance achieved through CPU partitioning only.
Low-latency	workloadHints: highPowerConsumption: false realTime: true	Regional datacenters	Both energy savings and low-latency are desirable: compromise between power management, latency and throughput.
Ultra-low-latency	workloadHints: highPowerConsumption: true realTime: true	Far edge clusters, latency critical workloads	Optimized for absolute minimal latency and maximum determinism at the cost of increased power consumption.
Per-pod power management	workloadHints: realTime: true highPowerConsumption: false perPodPowerManagement: true	Critical and non-critical workloads	Allows for power management per pod.

Additional resources

- For information about using the Performance Profile Creator (PPC) to generate a performance profile, see Creating a performance profile .

11.3.7. Configuring workload hints manually

Procedure

1. Create a **PerformanceProfile** appropriate for the environment's hardware and topology as described in the table in "Understanding workload hints". Adjust the profile to match the expected workload. In this example, we tune for the lowest possible latency.

2. Add the **highPowerConsumption** and **realTime** workload hints. Both are set to **true** here.

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: workload-hints
spec:
  ...
  workloadHints:
    highPowerConsumption: true 1
    realTime: true 2
```

1 If **highPowerConsumption** is **true**, the node is tuned for very low latency at the cost of increased power consumption.

2 Disables some debugging and monitoring features that can affect system latency.

> **NOTE**
>
> When the **realTime** workload hint flag is set to **true** in a performance profile, add the **cpu-quota.crio.io: disable** annotation to every guaranteed pod with pinned CPUs. This annotation is necessary to prevent the degradation of the process performance within the pod. If the **realTime** workload hint is not explicitly set then it defaults to **true**.

Additional resources

- For information about reducing CPU throttling for individual guaranteed pods, see Disabling CPU CFS quota.

11.3.8. Restricting CPUs for infra and application containers

Generic housekeeping and workload tasks use CPUs in a way that may impact latency-sensitive processes. By default, the container runtime uses all online CPUs to run all containers together, which can result in context switches and spikes in latency. Partitioning the CPUs prevents noisy processes from interfering with latency-sensitive processes by separating them from each other. The following table describes how processes run on a CPU after you have tuned the node using the Node Tuning Operator:

Table 11.2. Process' CPU assignments

Process type	Details
Burstable and **BestEffort** pods	Runs on any CPU except where low latency workload is running
Infrastructure pods	Runs on any CPU except where low latency workload is running
Interrupts	Redirects to reserved CPUs (optional in OpenShift Container Platform 4.7 and later)
Kernel processes	Pins to reserved CPUs
Latency-sensitive workload pods	Pins to a specific set of exclusive CPUs from the isolated pool
OS processes/systemd services	Pins to reserved CPUs

The allocatable capacity of cores on a node for pods of all QoS process types, **Burstable**, **BestEffort**, or **Guaranteed**, is equal to the capacity of the isolated pool. The capacity of the reserved pool is removed from the node's total core capacity for use by the cluster and operating system housekeeping duties.

Example 1

A node features a capacity of 100 cores. Using a performance profile, the cluster administrator allocates 50 cores to the isolated pool and 50 cores to the reserved pool. The cluster administrator assigns 25 cores to QoS **Guaranteed** pods and 25 cores for **BestEffort** or **Burstable** pods. This matches the capacity of the isolated pool.

Example 2

A node features a capacity of 100 cores. Using a performance profile, the cluster administrator allocates 50 cores to the isolated pool and 50 cores to the reserved pool. The cluster administrator assigns 50 cores to QoS **Guaranteed** pods and one core for **BestEffort** or **Burstable** pods. This exceeds the capacity of the isolated pool by one core. Pod scheduling fails because of insufficient CPU capacity.

The exact partitioning pattern to use depends on many factors like hardware, workload characteristics and the expected system load. Some sample use cases are as follows:

- If the latency-sensitive workload uses specific hardware, such as a network interface controller (NIC), ensure that the CPUs in the isolated pool are as close as possible to this hardware. At a minimum, you should place the workload in the same Non-Uniform Memory Access (NUMA) node.

- The reserved pool is used for handling all interrupts. When depending on system networking, allocate a sufficiently-sized reserve pool to handle all the incoming packet interrupts. In 4.15 and later versions, workloads can optionally be labeled as sensitive.

The decision regarding which specific CPUs should be used for reserved and isolated partitions requires detailed analysis and measurements. Factors like NUMA affinity of devices and memory play a role. The selection also depends on the workload architecture and the specific use case.

> **IMPORTANT**
>
> The reserved and isolated CPU pools must not overlap and together must span all available cores in the worker node.

To ensure that housekeeping tasks and workloads do not interfere with each other, specify two groups of CPUs in the **spec** section of the performance profile.

- **isolated** – Specifies the CPUs for the application container workloads. These CPUs have the lowest latency. Processes in this group have no interruptions and can, for example, reach much higher DPDK zero packet loss bandwidth.

- **reserved** – Specifies the CPUs for the cluster and operating system housekeeping duties. Threads in the **reserved** group are often busy. Do not run latency-sensitive applications in the **reserved** group. Latency-sensitive applications run in the **isolated** group.

Procedure

1. Create a performance profile appropriate for the environment's hardware and topology.

2. Add the **reserved** and **isolated** parameters with the CPUs you want reserved and isolated for the infra and application containers:

   ```
   apiVersion: performance.openshift.io/v2
   kind: PerformanceProfile
   metadata:
     name: infra-cpus
   spec:
     cpu:
       reserved: "0-4,9"   1
       isolated: "5-8"     2
     nodeSelector:         3
       node-role.kubernetes.io/worker: ""
   ```

   1 Specify which CPUs are for infra containers to perform cluster and operating system housekeeping duties.

   2 Specify which CPUs are for application containers to run workloads.

   3 Optional: Specify a node selector to apply the performance profile to specific nodes.

Additional resources

- Managing device interrupt processing for guaranteed pod isolated CPUs

- Create a pod that gets assigned a QoS class of Guaranteed

11.4. REDUCING NIC QUEUES USING THE NODE TUNING OPERATOR

The Node Tuning Operator allows you to adjust the network interface controller (NIC) queue count for each network device. By using a PerformanceProfile, the amount of queues can be reduced to the number of reserved CPUs.

11.4.1. Adjusting the NIC queues with the performance profile

The performance profile lets you adjust the queue count for each network device.

Supported network devices:

- Non-virtual network devices
- Network devices that support multiple queues (channels)

Unsupported network devices:

- Pure software network interfaces
- Block devices
- Intel DPDK virtual functions

Prerequisites

- Access to the cluster as a user with the **cluster-admin** role.
- Install the OpenShift CLI (**oc**).

Procedure

1. Log in to the OpenShift Container Platform cluster running the Node Tuning Operator as a user with **cluster-admin** privileges.

2. Create and apply a performance profile appropriate for your hardware and topology. For guidance on creating a profile, see the "Creating a performance profile" section.

3. Edit this created performance profile:

   ```
   $ oc edit -f <your_profile_name>.yaml
   ```

4. Populate the **spec** field with the **net** object. The object list can contain two fields:

   - **userLevelNetworking** is a required field specified as a boolean flag. If **userLevelNetworking** is **true**, the queue count is set to the reserved CPU count for all supported devices. The default is **false**.

   - **devices** is an optional field specifying a list of devices that will have the queues set to the reserved CPU count. If the device list is empty, the configuration applies to all network devices. The configuration is as follows:

     - **interfaceName**: This field specifies the interface name, and it supports shell-style wildcards, which can be positive or negative.

       - Example wildcard syntax is as follows: **<string> .***

       - Negative rules are prefixed with an exclamation mark. To apply the net queue changes to all devices other than the excluded list, use **!<device>**, for example, **!eno1**.

     - **vendorID**: The network device vendor ID represented as a 16-bit hexadecimal number with a **0x** prefix.

- **deviceID**: The network device ID (model) represented as a 16-bit hexadecimal number with a **0x** prefix.

> **NOTE**
>
> When a **deviceID** is specified, the **vendorID** must also be defined. A device that matches all of the device identifiers specified in a device entry **interfaceName**, **vendorID**, or a pair of **vendorID** plus **deviceID** qualifies as a network device. This network device then has its net queues count set to the reserved CPU count.
>
> When two or more devices are specified, the net queues count is set to any net device that matches one of them.

5. Set the queue count to the reserved CPU count for all devices by using this example performance profile:

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: manual
spec:
  cpu:
    isolated: 3-51,55-103
    reserved: 0-2,52-54
  net:
    userLevelNetworking: true
  nodeSelector:
    node-role.kubernetes.io/worker-cnf: ""
```

6. Set the queue count to the reserved CPU count for all devices matching any of the defined device identifiers by using this example performance profile:

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: manual
spec:
  cpu:
    isolated: 3-51,55-103
    reserved: 0-2,52-54
  net:
    userLevelNetworking: true
    devices:
    - interfaceName: "eth0"
    - interfaceName: "eth1"
    - vendorID: "0x1af4"
      deviceID: "0x1000"
  nodeSelector:
    node-role.kubernetes.io/worker-cnf: ""
```

7. Set the queue count to the reserved CPU count for all devices starting with the interface name **eth** by using this example performance profile:

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: manual
spec:
  cpu:
    isolated: 3-51,55-103
    reserved: 0-2,52-54
  net:
    userLevelNetworking: true
    devices:
    - interfaceName: "eth*"
  nodeSelector:
    node-role.kubernetes.io/worker-cnf: ""
```

8. Set the queue count to the reserved CPU count for all devices with an interface named anything other than **eno1** by using this example performance profile:

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: manual
spec:
  cpu:
    isolated: 3-51,55-103
    reserved: 0-2,52-54
  net:
    userLevelNetworking: true
    devices:
    - interfaceName: "!eno1"
  nodeSelector:
    node-role.kubernetes.io/worker-cnf: ""
```

9. Set the queue count to the reserved CPU count for all devices that have an interface name **eth0**, **vendorID** of **0x1af4**, and **deviceID** of **0x1000** by using this example performance profile:

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: manual
spec:
  cpu:
    isolated: 3-51,55-103
    reserved: 0-2,52-54
  net:
    userLevelNetworking: true
    devices:
    - interfaceName: "eth0"
    - vendorID: "0x1af4"
      deviceID: "0x1000"
  nodeSelector:
    node-role.kubernetes.io/worker-cnf: ""
```

10. Apply the updated performance profile:

```
$ oc apply -f <your_profile_name>.yaml
```

Additional resources

- [Creating a performance profile](#) .

11.4.2. Verifying the queue status

In this section, a number of examples illustrate different performance profiles and how to verify the changes are applied.

Example 1

In this example, the net queue count is set to the reserved CPU count (2) for *all* supported devices.

The relevant section from the performance profile is:

```
apiVersion: performance.openshift.io/v2
metadata:
  name: performance
spec:
  kind: PerformanceProfile
  spec:
    cpu:
      reserved: 0-1   #total = 2
      isolated: 2-8
    net:
      userLevelNetworking: true
# ...
```

- Display the status of the queues associated with a device using the following command:

  > **NOTE**
  >
  > Run this command on the node where the performance profile was applied.

  ```
  $ ethtool -l <device>
  ```

- Verify the queue status before the profile is applied:

  ```
  $ ethtool -l ens4
  ```

  Example output

  ```
  Channel parameters for ens4:
  Pre-set maximums:
  RX:        0
  TX:        0
  Other:     0
  Combined:  4
  Current hardware settings:
  RX:        0
  ```

```
TX:        0
Other:     0
Combined:  4
```

- Verify the queue status after the profile is applied:

```
$ ethtool -l ens4
```

Example output

```
Channel parameters for ens4:
Pre-set maximums:
RX:        0
TX:        0
Other:     0
Combined:  4
Current hardware settings:
RX:        0
TX:        0
Other:     0
Combined:  2
```
1

1   The combined channel shows that the total count of reserved CPUs for *all* supported devices is 2. This matches what is configured in the performance profile.

Example 2

In this example, the net queue count is set to the reserved CPU count (2) for *all* supported network devices with a specific **vendorID**.

The relevant section from the performance profile is:

```
apiVersion: performance.openshift.io/v2
metadata:
  name: performance
spec:
  kind: PerformanceProfile
  spec:
    cpu:
      reserved: 0-1  #total = 2
      isolated: 2-8
    net:
      userLevelNetworking: true
      devices:
      - vendorID = 0x1af4
# ...
```

- Display the status of the queues associated with a device using the following command:

> **NOTE**
>
> Run this command on the node where the performance profile was applied.

```
$ ethtool -l <device>
```

- Verify the queue status after the profile is applied:

```
$ ethtool -l ens4
```

Example output

```
Channel parameters for ens4:
Pre-set maximums:
RX:         0
TX:         0
Other:      0
Combined:   4
Current hardware settings:
RX:         0
TX:         0
Other:      0
Combined:   2  ❶
```

❶ The total count of reserved CPUs for all supported devices with **vendorID=0x1af4** is 2. For example, if there is another network device **ens2** with **vendorID=0x1af4** it will also have total net queues of 2. This matches what is configured in the performance profile.
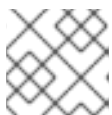
Example 3

In this example, the net queue count is set to the reserved CPU count (2) for *all* supported network devices that match any of the defined device identifiers.

The command **udevadm info** provides a detailed report on a device. In this example the devices are:

```
# udevadm info -p /sys/class/net/ens4
...
E: ID_MODEL_ID=0x1000
E: ID_VENDOR_ID=0x1af4
E: INTERFACE=ens4
...
```

```
# udevadm info -p /sys/class/net/eth0
...
E: ID_MODEL_ID=0x1002
E: ID_VENDOR_ID=0x1001
E: INTERFACE=eth0
...
```

- Set the net queues to 2 for a device with **interfaceName** equal to **eth0** and any devices that have a **vendorID=0x1af4** with the following performance profile:

```
apiVersion: performance.openshift.io/v2
metadata:
  name: performance
spec:
  kind: PerformanceProfile
```

```
    spec:
      cpu:
        reserved: 0-1  #total = 2
        isolated: 2-8
      net:
        userLevelNetworking: true
        devices:
        - interfaceName = eth0
        - vendorID = 0x1af4
    ...
```

- Verify the queue status after the profile is applied:

```
$ ethtool -l ens4
```

Example output

```
Channel parameters for ens4:
Pre-set maximums:
RX:        0
TX:        0
Other:     0
Combined:  4
Current hardware settings:
RX:        0
TX:        0
Other:     0
Combined:  2  1
```

1 The total count of reserved CPUs for all supported devices with **vendorID=0x1af4** is set to 2. For example, if there is another network device **ens2** with **vendorID=0x1af4**, it will also have the total net queues set to 2. Similarly, a device with **interfaceName** equal to **eth0** will have total net queues set to 2.

11.4.3. Logging associated with adjusting NIC queues

Log messages detailing the assigned devices are recorded in the respective Tuned daemon logs. The following messages might be recorded to the **/var/log/tuned/tuned.log** file:

- An **INFO** message is recorded detailing the successfully assigned devices:

```
INFO tuned.plugins.base: instance net_test (net): assigning devices ens1, ens2, ens3
```

- A **WARNING** message is recorded if none of the devices can be assigned:

```
WARNING  tuned.plugins.base: instance net_test: no matching devices available
```

11.5. DEBUGGING LOW LATENCY CNF TUNING STATUS

The **PerformanceProfile** custom resource (CR) contains status fields for reporting tuning status and debugging latency degradation issues. These fields report on conditions that describe the state of the operator's reconciliation functionality.

A typical issue can arise when the status of machine config pools that are attached to the performance profile are in a degraded state, causing the **PerformanceProfile** status to degrade. In this case, the machine config pool issues a failure message.

The Node Tuning Operator contains the **performanceProfile.spec.status.Conditions** status field:

```
Status:
  Conditions:
    Last Heartbeat Time:   2020-06-02T10:01:24Z
    Last Transition Time:  2020-06-02T10:01:24Z
    Status:           True
    Type:             Available
    Last Heartbeat Time:   2020-06-02T10:01:24Z
    Last Transition Time:  2020-06-02T10:01:24Z
    Status:           True
    Type:             Upgradeable
    Last Heartbeat Time:   2020-06-02T10:01:24Z
    Last Transition Time:  2020-06-02T10:01:24Z
    Status:           False
    Type:             Progressing
    Last Heartbeat Time:   2020-06-02T10:01:24Z
    Last Transition Time:  2020-06-02T10:01:24Z
    Status:           False
    Type:             Degraded
```

The **Status** field contains **Conditions** that specify **Type** values that indicate the status of the performance profile:

Available

All machine configs and Tuned profiles have been created successfully and are available for cluster components are responsible to process them (NTO, MCO, Kubelet).

Upgradeable

Indicates whether the resources maintained by the Operator are in a state that is safe to upgrade.

Progressing

Indicates that the deployment process from the performance profile has started.

Degraded

Indicates an error if:

- Validation of the performance profile has failed.

- Creation of all relevant components did not complete successfully.

Each of these types contain the following fields:

Status

The state for the specific type (**true** or **false**).

Timestamp

The transaction timestamp.

Reason string

The machine readable reason.

Message string

The human readable reason describing the state and error details, if any.

11.5.1. Machine config pools

A performance profile and its created products are applied to a node according to an associated machine config pool (MCP). The MCP holds valuable information about the progress of applying the machine configurations created by performance profiles that encompass kernel args, kube config, huge pages allocation, and deployment of rt-kernel. The Performance Profile controller monitors changes in the MCP and updates the performance profile status accordingly.

The only conditions returned by the MCP to the performance profile status is when the MCP is **Degraded**, which leads to **performanceProfile.status.condition.Degraded = true**.

Example

The following example is for a performance profile with an associated machine config pool (**worker-cnf**) that was created for it:

1. The associated machine config pool is in a degraded state:

   ```
   # oc get mcp
   ```

   Example output

   ```
   NAME        CONFIG                                         UPDATED  UPDATING  DEGRADED
   MACHINECOUNT   READYMACHINECOUNT   UPDATEDMACHINECOUNT
   DEGRADEDMACHINECOUNT   AGE
   master      rendered-master-2ee57a93fa6c9181b546ca46e1571d2d    True    False
   False    3       3       3          0           2d21h
   worker      rendered-worker-d6b2bdc07d9f5a59a6b68950acf25e5f    True    False
   False    2       2       2          0           2d21h
   worker-cnf  rendered-worker-cnf-6c838641b8a08fff08dbd8b02fb63f7c  False    True
   True    2       1       1          1           2d20h
   ```

2. The **describe** section of the MCP shows the reason:

   ```
   # oc describe mcp worker-cnf
   ```

   Example output

   ```
     Message:            Node node-worker-cnf is reporting: "prepping update:
     machineconfig.machineconfiguration.openshift.io \"rendered-worker-cnf-
   40b9996919c08e335f3ff230ce1d170\" not
     found"
       Reason:            1 nodes are reporting degraded status on sync
   ```

3. The degraded state should also appear under the performance profile **status** field marked as **degraded = true**:

   ```
   # oc describe performanceprofiles performance
   ```

   Example output

> Message: Machine config pool worker-cnf Degraded Reason: 1 nodes are reporting
> degraded status on sync.
> Machine config pool worker-cnf Degraded Message: Node yquinn-q8s5v-w-b-
> z5lqn.c.openshift-gce-devel.internal is
> reporting: "prepping update: machineconfig.machineconfiguration.openshift.io
> \"rendered-worker-cnf-40b9996919c08e335f3ff230ce1d170\" not found".   Reason:
> MCPDegraded
>   Status: True
>   Type: Degraded

11.6. COLLECTING LOW LATENCY TUNING DEBUGGING DATA FOR RED HAT SUPPORT

When opening a support case, it is helpful to provide debugging information about your cluster to Red Hat Support.

The **must-gather** tool enables you to collect diagnostic information about your OpenShift Container Platform cluster, including node tuning, NUMA topology, and other information needed to debug issues with low latency setup.

For prompt support, supply diagnostic information for both OpenShift Container Platform and low latency tuning.

11.6.1. About the must-gather tool

The **oc adm must-gather** CLI command collects the information from your cluster that is most likely needed for debugging issues, such as:

- Resource definitions

- Audit logs

- Service logs

You can specify one or more images when you run the command by including the **--image** argument. When you specify an image, the tool collects data related to that feature or product. When you run **oc adm must-gather**, a new pod is created on the cluster. The data is collected on that pod and saved in a new directory that starts with **must-gather.local**. This directory is created in your current working directory.

11.6.2. Gathering low latency tuning data

Use the **oc adm must-gather** CLI command to collect information about your cluster, including features and objects associated with low latency tuning, including:

- The Node Tuning Operator namespaces and child objects.

- **MachineConfigPool** and associated **MachineConfig** objects.

- The Node Tuning Operator and associated Tuned objects.

- Linux kernel command line options.

- CPU and NUMA topology

- Basic PCI device information and NUMA locality.

Prerequisites

- Access to the cluster as a user with the **cluster-admin** role.

- The OpenShift Container Platform CLI (oc) installed.

Procedure

1. Navigate to the directory where you want to store the **must-gather** data.

2. Collect debugging information by running the following command:

```
$ oc adm must-gather
```

Example output

```
[must-gather      ] OUT Using must-gather plug-in image: quay.io/openshift-release
When opening a support case, bugzilla, or issue please include the following summary data
along with any other requested information:
ClusterID: 829er0fa-1ad8-4e59-a46e-2644921b7eb6
ClusterVersion: Stable at "<cluster_version>"
ClusterOperators:
 All healthy and stable


[must-gather      ] OUT namespace/openshift-must-gather-8fh4x created
[must-gather      ] OUT clusterrolebinding.rbac.authorization.k8s.io/must-gather-rhlgc created
[must-gather-5564g] POD 2023-07-17T10:17:37.610340849Z Gathering data for
ns/openshift-cluster-version...
[must-gather-5564g] POD 2023-07-17T10:17:38.786591298Z Gathering data for ns/default...
[must-gather-5564g] POD 2023-07-17T10:17:39.117418660Z Gathering data for
ns/openshift...
[must-gather-5564g] POD 2023-07-17T10:17:39.447592859Z Gathering data for ns/kube-
system...
[must-gather-5564g] POD 2023-07-17T10:17:39.803381143Z Gathering data for
ns/openshift-etcd...


...


Reprinting Cluster State:
When opening a support case, bugzilla, or issue please include the following summary data
along with any other requested information:
ClusterID: 829er0fa-1ad8-4e59-a46e-2644921b7eb6
ClusterVersion: Stable at "<cluster_version>"
ClusterOperators:
 All healthy and stable
```

3. Create a compressed file from the **must-gather** directory that was created in your working directory. For example, on a computer that uses a Linux operating system, run the following command:

```
$ tar cvaf must-gather.tar.gz must-gather-local.5421342344627712289
```

[1] Replace **must-gather-local.5421342344627712289**// with the directory name created by the **must-gather** tool.
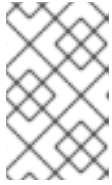
> **NOTE**
>
> Create a compressed file to attach the data to a support case or to use with the Performance Profile Creator wrapper script when you create a performance profile.

4. Attach the compressed file to your support case on the Red Hat Customer Portal .

Additional resources

- For more information about the **must-gather** tool, see Gathering data about your cluster

- For more information about MachineConfig and KubeletConfig, see Managing nodes.

- For more information about the Node Tuning Operator, see Using the Node Tuning Operator .

- For more information about the PerformanceProfile, see Configuring huge pages .

- For more information about consuming huge pages from your containers, see How huge pages are consumed by apps.

CHAPTER 12. PERFORMING LATENCY TESTS FOR PLATFORM VERIFICATION

You can use the Cloud-native Network Functions (CNF) tests image to run latency tests on a CNF-enabled OpenShift Container Platform cluster, where all the components required for running CNF workloads are installed. Run the latency tests to validate node tuning for your workload.

The **cnf-tests** container image is available at **registry.redhat.io/openshift4/cnf-tests-rhel8:v4.15**.

12.1. PREREQUISITES FOR RUNNING LATENCY TESTS

Your cluster must meet the following requirements before you can run the latency tests:

1. You have configured a performance profile with the Node Tuning Operator.

2. You have applied all the required CNF configurations in the cluster.

3. You have a pre-existing **MachineConfigPool** CR applied in the cluster. The default worker pool is **worker-cnf**.

Additional resources

- For more information about creating the cluster performance profile, see Provisioning a worker with real-time capabilities.

12.2. MEASURING LATENCY

The **cnf-tests** image uses three tools to measure the latency of the system:

- **hwlatdetect**

- **cyclictest**

- **oslat**

Each tool has a specific use. Use the tools in sequence to achieve reliable test results.

hwlatdetect

Measures the baseline that the bare-metal hardware can achieve. Before proceeding with the next latency test, ensure that the latency reported by **hwlatdetect** meets the required threshold because you cannot fix hardware latency spikes by operating system tuning.

cyclictest

Verifies the real-time kernel scheduler latency after **hwlatdetect** passes validation. The **cyclictest** tool schedules a repeated timer and measures the difference between the desired and the actual trigger times. The difference can uncover basic issues with the tuning caused by interrupts or process priorities. The tool must run on a real-time kernel.

oslat

Behaves similarly to a CPU-intensive DPDK application and measures all the interruptions and disruptions to the busy loop that simulates CPU heavy data processing.

The tests introduce the following environment variables:

Table 12.1. Latency test environment variables

Environment variables	Description
LATENCY_TEST_DELAY	Specifies the amount of time in seconds after which the test starts running. You can use the variable to allow the CPU manager reconcile loop to update the default CPU pool. The default value is 0.
LATENCY_TEST_CPUS	Specifies the number of CPUs that the pod running the latency tests uses. If you do not set the variable, the default configuration includes all isolated CPUs.
LATENCY_TEST_RUNTIME	Specifies the amount of time in seconds that the latency test must run. The default value is 300 seconds. **NOTE** To prevent the Ginkgo 2.0 test suite from timing out before the latency tests complete, set the **-ginkgo.timeout** flag to a value greater than **LATENCY_TEST_RUNTIME** + 2 minutes. If you also set a **LATENCY_TEST_DELAY** value then you must set **-ginkgo.timeout** to a value greater than **LATENCY_TEST_RUNTIME** + **LATENCY_TEST_DELAY** + 2 minutes. The default timeout value for the Ginkgo 2.0 test suite is 1 hour.
HWLATDETECT_MAXIMUM_LATENCY	Specifies the maximum acceptable hardware latency in microseconds for the workload and operating system. If you do not set the value of **HWLATDETECT_MAXIMUM_LATENCY** or **MAXIMUM_LATENCY**, the tool compares the default expected threshold (20μs) and the actual maximum latency in the tool itself. Then, the test fails or succeeds accordingly.
CYCLICTEST_MAXIMUM_LATENCY	Specifies the maximum latency in microseconds that all threads expect before waking up during the **cyclictest** run. If you do not set the value of **CYCLICTEST_MAXIMUM_LATENCY** or **MAXIMUM_LATENCY**, the tool skips the comparison of the expected and the actual maximum latency.
OSLAT_MAXIMUM_LATENCY	Specifies the maximum acceptable latency in microseconds for the **oslat** test results. If you do not set the value of **OSLAT_MAXIMUM_LATENCY** or **MAXIMUM_LATENCY**, the tool skips the comparison of the expected and the actual maximum latency.
MAXIMUM_LATENCY	Unified variable that specifies the maximum acceptable latency in microseconds. Applicable for all available latency tools.

NOTE

Variables that are specific to a latency tool take precedence over unified variables. For example, if **OSLAT_MAXIMUM_LATENCY** is set to 30 microseconds and **MAXIMUM_LATENCY** is set to 10 microseconds, the **oslat** test will run with maximum acceptable latency of 30 microseconds.

12.3. RUNNING THE LATENCY TESTS

Run the cluster latency tests to validate node tuning for your Cloud-native Network Functions (CNF) workload.

> **NOTE**
>
> When executing **podman** commands as a non-root or non-privileged user, mounting paths can fail with **permission denied** errors. To make the **podman** command work, append **:Z** to the volumes creation; for example, **-v $(pwd)/:/kubeconfig:Z**. This allows **podman** to do the proper SELinux relabeling.

Procedure

1. Open a shell prompt in the directory containing the **kubeconfig** file.
   You provide the test image with a **kubeconfig** file in current directory and its related **$KUBECONFIG** environment variable, mounted through a volume. This allows the running container to use the **kubeconfig** file from inside the container.

2. Run the latency tests by entering the following command:

   ```
   $ podman run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig \
   -e LATENCY_TEST_RUNTIME=<time_in_seconds>\
   -e MAXIMUM_LATENCY=<time_in_microseconds> \
   registry.redhat.io/openshift4/cnf-tests-rhel8:v4.15 /usr/bin/test-run.sh \
   --ginkgo.v --ginkgo.timeout="24h"
   ```

3. Optional: Append **--ginkgo.dryRun** flag to run the latency tests in dry-run mode. This is useful for checking what commands the tests run.

4. Optional: Append **--ginkgo.v** flag to run the tests with increased verbosity.

5. Optional: Append **--ginkgo.timeout="24h"** flag to ensure the Ginkgo 2.0 test suite does not timeout before the latency tests complete.

> **IMPORTANT**
>
> The default runtime for each test is 300 seconds. For valid latency test results, run the tests for at least 12 hours by updating the **LATENCY_TEST_RUNTIME** variable.

12.3.1. Running hwlatdetect

The **hwlatdetect** tool is available in the **rt-kernel** package with a regular subscription of Red Hat Enterprise Linux (RHEL) 9.x.

> **NOTE**
>
> When executing **podman** commands as a non-root or non-privileged user, mounting paths can fail with **permission denied** errors. To make the **podman** command work, append **:Z** to the volumes creation; for example, **-v $(pwd)/:/kubeconfig:Z**. This allows **podman** to do the proper SELinux relabeling.

Prerequisites

- You have installed the real-time kernel in the cluster.

- You have logged in to **registry.redhat.io** with your Customer Portal credentials.

Procedure

- To run the **hwlatdetect** tests, run the following command, substituting variable values as appropriate:

```
$ podman run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig \
-e LATENCY_TEST_RUNTIME=600 -e MAXIMUM_LATENCY=20 \
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.15 \
/usr/bin/test-run.sh --ginkgo.focus="hwlatdetect" --ginkgo.v --ginkgo.timeout="24h"
```

The **hwlatdetect** test runs for 10 minutes (600 seconds). The test runs successfully when the maximum observed latency is lower than **MAXIMUM_LATENCY** (20 μs).

If the results exceed the latency threshold, the test fails.
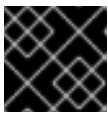
> **IMPORTANT**
>
> For valid results, the test should run for at least 12 hours.

Example failure output

```
running /usr/bin/cnftests -ginkgo.v -ginkgo.focus=hwlatdetect
I0908 15:25:20.023712      27 request.go:601] Waited for 1.046586367s due to client-side
throttling, not priority and fairness, request:
GET:https://api.hlxcl6.lab.eng.tlv2.redhat.com:6443/apis/imageregistry.operator.openshift.io/v1?
timeout=32s
Running Suite: CNF Features e2e integration tests
==================================================
Random Seed: 1662650718
Will run 1 of 3 specs

[...]

• Failure [283.574 seconds]
[performance] Latency Test
/remote-source/app/vendor/github.com/openshift/cluster-node-tuning-
operator/test/e2e/performanceprofile/functests/4_latency/latency.go:62
  with the hwlatdetect image
  /remote-source/app/vendor/github.com/openshift/cluster-node-tuning-
operator/test/e2e/performanceprofile/functests/4_latency/latency.go:228
    should succeed [It]
    /remote-source/app/vendor/github.com/openshift/cluster-node-tuning-
operator/test/e2e/performanceprofile/functests/4_latency/latency.go:236

    Log file created at: 2022/09/08 15:25:27
    Running on machine: hwlatdetect-b6n4n
    Binary: Built with gc go1.17.12 for linux/amd64
    Log line format: [IWEF]mmdd hh:mm:ss.uuuuuu threadid file:line] msg
    I0908 15:25:27.160620      1 node.go:39] Environment information: /proc/cmdline:
BOOT_IMAGE=(hd1,gpt3)/ostree/rhcos-
c6491e1eedf6c1f12ef7b95e14ee720bf48359750ac900b7863c625769ef5fb9/vmlinuz-4.18.0-
372.19.1.el8_6.x86_64 random.trust_cpu=on console=tty0 console=ttyS0,115200n8
ignition.platform.id=metal
```

```
ostree=/ostree/boot.1/rhcos/c6491e1eedf6c1f12ef7b95e14ee720bf48359750ac900b7863c625
769ef5fb9/0 ip=dhcp root=UUID=5f80c283-f6e6-4a27-9b47-a287157483b2 rw
rootflags=prjquota boot=UUID=773bf59a-bafd-48fc-9a87-f62252d739d3 skew_tick=1
nohz=on rcu_nocbs=0-3 tuned.non_isolcpus=0000ffff,ffffffff,ffffffff0
systemd.cpu_affinity=4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29
,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,
60,61,62,63,64,65,66,67,68,69,70,71,72,73,74,75,76,77,78,79 intel_iommu=on iommu=pt
isolcpus=managed_irq,0-3 nohz_full=0-3 tsc=nowatchdog nosoftlockup nmi_watchdog=0
mce=off skew_tick=1 rcutree.kthread_prio=11 + +
    I0908 15:25:27.160830       1 node.go:46] Environment information: kernel version 4.18.0-
372.19.1.el8_6.x86_64
    I0908 15:25:27.160857       1 main.go:50] running the hwlatdetect command with
arguments [/usr/bin/hwlatdetect --threshold 1 --hardlimit 1 --duration 100 --window
10000000us --width 950000us]
    F0908 15:27:10.603523       1 main.go:53] failed to run hwlatdetect command; out:
hwlatdetect:  test duration 100 seconds
      detector: tracer
      parameters:
        Latency threshold: 1us
        Sample window:     10000000us
        Sample width:       950000us
      Non-sampling period:  9050000us
        Output File:      None

    Starting test
    test finished
    Max Latency: 326us
    Samples recorded: 5
    Samples exceeding threshold: 5
    ts: 1662650739.017274507, inner:6, outer:6
    ts: 1662650749.257272414, inner:14, outer:326
    ts: 1662650779.977272835, inner:314, outer:12
    ts: 1662650800.457272384, inner:3, outer:9
    ts: 1662650810.697273520, inner:3, outer:2

[...]

JUnit report was created: /junit.xml/cnftests-junit.xml


Summarizing 1 Failure:

[Fail] [performance] Latency Test with the hwlatdetect image [It] should succeed
/remote-source/app/vendor/github.com/openshift/cluster-node-tuning-
operator/test/e2e/performanceprofile/functests/4_latency/latency.go:476

Ran 1 of 194 Specs in 365.797 seconds
FAIL! -- 0 Passed | 1 Failed | 0 Pending | 2 Skipped
--- FAIL: TestTest (366.08s)
FAIL
```

1 You can configure the latency threshold by using the **MAXIMUM_LATENCY** or the **HWLATDETECT_MAXIMUM_LATENCY** environment variables.

2 The maximum latency value measured during the test.

Example hwlatdetect test results

You can capture the following types of results:

- Rough results that are gathered after each run to create a history of impact on any changes made throughout the test.

- The combined set of the rough tests with the best results and configuration settings.

Example of good results

```
hwlatdetect: test duration 3600 seconds
detector: tracer
parameters:
Latency threshold: 10us
Sample window: 1000000us
Sample width: 950000us
Non-sampling period: 50000us
Output File: None

Starting test
test finished
Max Latency: Below threshold
Samples recorded: 0
```

The **hwlatdetect** tool only provides output if the sample exceeds the specified threshold.

Example of bad results

```
hwlatdetect: test duration 3600 seconds
detector: tracer
parameters:Latency threshold: 10usSample window: 1000000us
Sample width: 950000usNon-sampling period: 50000usOutput File: None

Starting tests:1610542421.275784439, inner:78, outer:81
ts: 1610542444.330561619, inner:27, outer:28
ts: 1610542445.332549975, inner:39, outer:38
ts: 1610542541.568546097, inner:47, outer:32
ts: 1610542590.681548531, inner:13, outer:17
ts: 1610543033.818801482, inner:29, outer:30
ts: 1610543080.938801990, inner:90, outer:76
ts: 1610543129.065549639, inner:28, outer:39
ts: 1610543474.859552115, inner:28, outer:35
ts: 1610543523.973856571, inner:52, outer:49
ts: 1610543572.089799738, inner:27, outer:30
ts: 1610543573.091550771, inner:34, outer:28
ts: 1610543574.093555202, inner:116, outer:63
```

The output of **hwlatdetect** shows that multiple samples exceed the threshold. However, the same output can indicate different results based on the following factors:

- The duration of the test

- The number of CPU cores

- The host firmware settings

> **WARNING**
>
> Before proceeding with the next latency test, ensure that the latency reported by **hwlatdetect** meets the required threshold. Fixing latencies introduced by hardware might require you to contact the system vendor support.
>
> Not all latency spikes are hardware related. Ensure that you tune the host firmware to meet your workload requirements. For more information, see Setting firmware parameters for system tuning.

12.3.2. Running cyclictest

The **cyclictest** tool measures the real-time kernel scheduler latency on the specified CPUs.
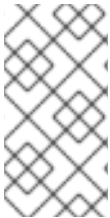
> **NOTE**
>
> When executing **podman** commands as a non-root or non-privileged user, mounting paths can fail with **permission denied** errors. To make the **podman** command work, append **:Z** to the volumes creation; for example, **-v $(pwd)/:/kubeconfig:Z**. This allows **podman** to do the proper SELinux relabeling.

Prerequisites

- You have logged in to **registry.redhat.io** with your Customer Portal credentials.

- You have installed the real-time kernel in the cluster.

- You have applied a cluster performance profile by using Node Tuning Operator.

Procedure

- To perform the **cyclictest**, run the following command, substituting variable values as appropriate:

```
$ podman run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig \
-e LATENCY_TEST_CPUS=10 -e LATENCY_TEST_RUNTIME=600 -e
MAXIMUM_LATENCY=20 \
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.15 \
/usr/bin/test-run.sh --ginkgo.focus="cyclictest" --ginkgo.v --ginkgo.timeout="24h"
```

The command runs the **cyclictest** tool for 10 minutes (600 seconds). The test runs successfully when the maximum observed latency is lower than **MAXIMUM_LATENCY** (in this example, 20 μs). Latency spikes of 20 μs and above are generally not acceptable for {rds} workloads.

If the results exceed the latency threshold, the test fails.
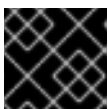
> **IMPORTANT**
>
> For valid results, the test should run for at least 12 hours.

Example failure output

```
running /usr/bin/cnftests -ginkgo.v -ginkgo.focus=cyclictest
I0908 13:01:59.193776      27 request.go:601] Waited for 1.046228824s due to client-side
throttling, not priority and fairness, request: GET:https://api.compute-
1.example.com:6443/apis/packages.operators.coreos.com/v1?timeout=32s
Running Suite: CNF Features e2e integration tests
=====================================================
Random Seed: 1662642118
Will run 1 of 3 specs

[...]

Summarizing 1 Failure:

[Fail] [performance] Latency Test with the cyclictest image [It] should succeed
/remote-source/app/vendor/github.com/openshift/cluster-node-tuning-
operator/test/e2e/performanceprofile/functests/4_latency/latency.go:220

Ran 1 of 194 Specs in 161.151 seconds
FAIL! -- 0 Passed | 1 Failed | 0 Pending | 2 Skipped
--- FAIL: TestTest (161.48s)
FAIL
```

Example cyclictest results

The same output can indicate different results for different workloads. For example, spikes up to 18μs are acceptable for 4G DU workloads, but not for 5G DU workloads.

Example of good results

```
running cmd: cyclictest -q -D 10m -p 1 -t 16 -a 2,4,6,8,10,12,14,16,54,56,58,60,62,64,66,68 -h 30 -i
1000 -m
# Histogram
000000 000000   000000  000000  000000  000000  000000  000000  000000  000000  000000
000000  000000  000000  000000  000000  000000
000001 000000   000000  000000  000000  000000  000000  000000  000000  000000  000000
000000  000000  000000  000000  000000  000000
000002 579506   535967  418614  573648  532870  529897  489306  558076  582350  585188
583793  223781  532480  569130  472250  576043
More histogram entries ...
# Total: 000600000 000600000 000600000 000599999 000599999 000599999 000599998
000599998 000599998 000599997 000599997 000599996 000599996 000599995 000599995
000599995
# Min Latencies: 00002 00002 00002 00002 00002 00002 00002 00002 00002 00002 00002 00002
00002 00002 00002 00002
# Avg Latencies: 00002 00002 00002 00002 00002 00002 00002 00002 00002 00002 00002 00002
00002 00002 00002 00002
# Max Latencies: 00005 00005 00004 00005 00004 00004 00005 00005 00006 00005 00004 00005
00004 00004 00005 00004
# Histogram Overflows: 00000 00000 00000 00000 00000 00000 00000 00000 00000 00000 00000
00000 00000 00000 00000 00000
# Histogram Overflow at cycle number:
# Thread 0:
# Thread 1:
# Thread 2:
```

```
# Thread 3:
# Thread 4:
# Thread 5:
# Thread 6:
# Thread 7:
# Thread 8:
# Thread 9:
# Thread 10:
# Thread 11:
# Thread 12:
# Thread 13:
# Thread 14:
# Thread 15:
```

Example of bad results

```
running cmd: cyclictest -q -D 10m -p 1 -t 16 -a 2,4,6,8,10,12,14,16,54,56,58,60,62,64,66,68 -h 30 -i
1000 -m
# Histogram
000000 000000   000000 000000 000000 000000 000000 000000 000000 000000 000000
000000 000000 000000 000000 000000 000000
000001 000000   000000 000000 000000 000000 000000 000000 000000 000000 000000
000000 000000 000000 000000 000000 000000
000002 564632   579686 354911 563036 492543 521983 515884 378266 592621 463547
482764 591976 590409 588145 589556 353518
More histogram entries ...
# Total: 000599999 000599999 000599999 000599997 000599997 000599998 000599998
000599997 000599997 000599996 000599995 000599996 000599995 000599995 000599995
000599993
# Min Latencies: 00002 00002 00002 00002 00002 00002 00002 00002 00002 00002 00002 00002
00002 00002 00002 00002
# Avg Latencies: 00002 00002 00002 00002 00002 00002 00002 00002 00002 00002 00002 00002
00002 00002 00002 00002
# Max Latencies: 00493 00387 00271 00619 00541 00513 00009 00389 00252 00215 00539 00498
00363 00204 00068 00520
# Histogram Overflows: 00001 00001 00001 00002 00002 00001 00000 00001 00001 00001 00002
00001 00001 00001 00001 00002
# Histogram Overflow at cycle number:
# Thread 0: 155922
# Thread 1: 110064
# Thread 2: 110064
# Thread 3: 110063 155921
# Thread 4: 110063 155921
# Thread 5: 155920
# Thread 6:
# Thread 7: 110062
# Thread 8: 110062
# Thread 9: 155919
# Thread 10: 110061 155919
# Thread 11: 155918
# Thread 12: 155918
# Thread 13: 110060
# Thread 14: 110060
# Thread 15: 110059 155917
```

12.3.3. Running oslat

The **oslat** test simulates a CPU-intensive DPDK application and measures all the interruptions and disruptions to test how the cluster handles CPU heavy data processing.

> **NOTE**
>
> When executing **podman** commands as a non-root or non-privileged user, mounting paths can fail with **permission denied** errors. To make the **podman** command work, append **:Z** to the volumes creation; for example, **-v $(pwd)/:/kubeconfig:Z**. This allows **podman** to do the proper SELinux relabeling.

Prerequisites

- You have logged in to **registry.redhat.io** with your Customer Portal credentials.

- You have applied a cluster performance profile by using the Node Tuning Operator.

Procedure

- To perform the **oslat** test, run the following command, substituting variable values as appropriate:

```
$ podman run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig \
-e LATENCY_TEST_CPUS=10 -e LATENCY_TEST_RUNTIME=600 -e
MAXIMUM_LATENCY=20 \
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.15 \
/usr/bin/test-run.sh --ginkgo.focus="oslat" --ginkgo.v --ginkgo.timeout="24h"
```

LATENCY_TEST_CPUS specifies the number of CPUs to test with the **oslat** command.

The command runs the **oslat** tool for 10 minutes (600 seconds). The test runs successfully when the maximum observed latency is lower than **MAXIMUM_LATENCY** (20 μs).

If the results exceed the latency threshold, the test fails.
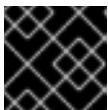
> **IMPORTANT**
>
> For valid results, the test should run for at least 12 hours.

Example failure output

```
running /usr/bin/cnftests -ginkgo.v -ginkgo.focus=oslat
I0908 12:51:55.999393      27 request.go:601] Waited for 1.044848101s due to client-side
throttling, not priority and fairness, request: GET:https://compute-
1.example.com:6443/apis/machineconfiguration.openshift.io/v1?timeout=32s
Running Suite: CNF Features e2e integration tests
=================================================
Random Seed: 1662641514
Will run 1 of 3 specs

[...]

• Failure [77.833 seconds]
```

```
[performance] Latency Test
/remote-source/app/vendor/github.com/openshift/cluster-node-tuning-
operator/test/e2e/performanceprofile/functests/4_latency/latency.go:62
  with the oslat image
  /remote-source/app/vendor/github.com/openshift/cluster-node-tuning-
operator/test/e2e/performanceprofile/functests/4_latency/latency.go:128
    should succeed [It]
    /remote-source/app/vendor/github.com/openshift/cluster-node-tuning-
operator/test/e2e/performanceprofile/functests/4_latency/latency.go:153

    The current latency 304 is bigger than the expected one 1 :
```


```
[...]

Summarizing 1 Failure:

[Fail] [performance] Latency Test with the oslat image [It] should succeed
/remote-source/app/vendor/github.com/openshift/cluster-node-tuning-
operator/test/e2e/performanceprofile/functests/4_latency/latency.go:177

Ran 1 of 194 Specs in 161.091 seconds
FAIL! -- 0 Passed | 1 Failed | 0 Pending | 2 Skipped
--- FAIL: TestTest (161.42s)
FAIL
```

 In this example, the measured latency is outside the maximum allowed value.

12.4. GENERATING A LATENCY TEST FAILURE REPORT

Use the following procedures to generate a JUnit latency test output and test failure report.

Prerequisites

- You have installed the OpenShift CLI (**oc**).

- You have logged in as a user with **cluster-admin** privileges.

Procedure

- Create a test failure report with information about the cluster state and resources for troubleshooting by passing the **--report** parameter with the path to where the report is dumped:

  ```
  $ podman run -v $(pwd)/:/kubeconfig:Z -v $(pwd)/reportdest:<report_folder_path> \
  -e KUBECONFIG=/kubeconfig/kubeconfig registry.redhat.io/openshift4/cnf-tests-rhel8:v4.15 \
  /usr/bin/test-run.sh --report <report_folder_path> --ginkgo.v
  ```

  where:

  <report_folder_path>
      Is the path to the folder where the report is generated.

12.5. GENERATING A JUNIT LATENCY TEST REPORT

Use the following procedures to generate a JUnit latency test output and test failure report.

Prerequisites

- You have installed the OpenShift CLI (**oc**).

- You have logged in as a user with **cluster-admin** privileges.

Procedure

- Create a JUnit-compliant XML report by passing the **--junit** parameter together with the path to where the report is dumped:

  > **NOTE**
  >
  > You must create the **junit** folder before running this command.

  ```
  $ podman run -v $(pwd)/:/kubeconfig:Z -v $(pwd)/junit:/junit \
  -e KUBECONFIG=/kubeconfig/kubeconfig registry.redhat.io/openshift4/cnf-tests-rhel8:v4.15 \
  /usr/bin/test-run.sh --ginkgo.junit-report junit/<file-name>.xml --ginkgo.v
  ```

  where:

  junit

  Is the folder where the junit report is stored.

12.6. RUNNING LATENCY TESTS ON A SINGLE-NODE OPENSHIFT CLUSTER

You can run latency tests on single-node OpenShift clusters.

> **NOTE**
>
> When executing **podman** commands as a non-root or non-privileged user, mounting paths can fail with **permission denied** errors. To make the **podman** command work, append **:Z** to the volumes creation; for example, **-v $(pwd)/:/kubeconfig:Z**. This allows **podman** to do the proper SELinux relabeling.

Prerequisites

- You have installed the OpenShift CLI (**oc**).

- You have logged in as a user with **cluster-admin** privileges.

- You have applied a cluster performance profile by using the Node Tuning Operator.

Procedure

- To run the latency tests on a single-node OpenShift cluster, run the following command:

  ```
  $ podman run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig \
  -e LATENCY_TEST_RUNTIME=<time_in_seconds> registry.redhat.io/openshift4/cnf-tests-
  ```

```
rhel8:v4.15 \
/usr/bin/test-run.sh --ginkgo.v --ginkgo.timeout="24h"
```

> **NOTE**
>
> The default runtime for each test is 300 seconds. For valid latency test results, run the tests for at least 12 hours by updating the **LATENCY_TEST_RUNTIME** variable. To run the buckets latency validation step, you must specify a maximum latency. For details on maximum latency variables, see the table in the "Measuring latency" section.

After running the test suite, all the dangling resources are cleaned up.

12.7. RUNNING LATENCY TESTS IN A DISCONNECTED CLUSTER

The CNF tests image can run tests in a disconnected cluster that is not able to reach external registries. This requires two steps:

1. Mirroring the **cnf-tests** image to the custom disconnected registry.

2. Instructing the tests to consume the images from the custom disconnected registry.

Mirroring the images to a custom registry accessible from the cluster
A **mirror** executable is shipped in the image to provide the input required by   **oc** to mirror the test image to a local registry.

1. Run this command from an intermediate machine that has access to the cluster and registry.redhat.io:

   ```
   $ podman run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig \
   registry.redhat.io/openshift4/cnf-tests-rhel8:v4.15 \
   /usr/bin/mirror -registry <disconnected_registry> | oc image mirror -f -
   ```

   where:

   <disconnected_registry>

   Is the disconnected mirror registry you have configured, for example, **my.local.registry:5000**/.

2. When you have mirrored the **cnf-tests** image into the disconnected registry, you must override the original registry used to fetch the images when running the tests, for example:

   ```
   podman run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig \
   -e IMAGE_REGISTRY="<disconnected_registry>" \
   -e CNF_TESTS_IMAGE="cnf-tests-rhel8:v4.15" \
   -e LATENCY_TEST_RUNTIME=<time_in_seconds> \
   <disconnected_registry>/cnf-tests-rhel8:v4.15 /usr/bin/test-run.sh --ginkgo.v --
   ginkgo.timeout="24h"
   ```

Configuring the tests to consume images from a custom registry
You can run the latency tests using a custom test image and image registry using **CNF_TESTS_IMAGE** and **IMAGE_REGISTRY** variables.

- To configure the latency tests to use a custom test image and image registry, run the following command:

```
$ podman run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig \
-e IMAGE_REGISTRY="<custom_image_registry>" \
-e CNF_TESTS_IMAGE="<custom_cnf-tests_image>" \
-e LATENCY_TEST_RUNTIME=<time_in_seconds> \
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.15 /usr/bin/test-run.sh --ginkgo.v --
ginkgo.timeout="24h"
```

where:

<custom_image_registry>

is the custom image registry, for example, **custom.registry:5000/**.

<custom_cnf-tests_image>

is the custom cnf-tests image, for example, **custom-cnf-tests-image:latest**.

Mirroring images to the cluster OpenShift image registry

OpenShift Container Platform provides a built-in container image registry, which runs as a standard workload on the cluster.

Procedure

1. Gain external access to the registry by exposing it with a route:

   ```
   $ oc patch configs.imageregistry.operator.openshift.io/cluster --patch '{"spec":
   {"defaultRoute":true}}' --type=merge
   ```

2. Fetch the registry endpoint by running the following command:

   ```
   $ REGISTRY=$(oc get route default-route -n openshift-image-registry --template='{{
   .spec.host }}')
   ```

3. Create a namespace for exposing the images:

   ```
   $ oc create ns cnftests
   ```

4. Make the image stream available to all the namespaces used for tests. This is required to allow the tests namespaces to fetch the images from the **cnf-tests** image stream. Run the following commands:

   ```
   $ oc policy add-role-to-user system:image-puller system:serviceaccount:cnf-features-
   testing:default --namespace=cnftests
   ```

   ```
   $ oc policy add-role-to-user system:image-puller system:serviceaccount:performance-addon-
   operators-testing:default --namespace=cnftests
   ```

5. Retrieve the docker secret name and auth token by running the following commands:

   ```
   $ SECRET=$(oc -n cnftests get secret | grep builder-docker | awk {'print $1'}
   ```

```
$ TOKEN=$(oc -n cnftests get secret $SECRET -o jsonpath="{.data['\.dockercfg']}" | base64
--decode | jq '.["image-registry.openshift-image-registry.svc:5000"].auth')
```

6. Create a **dockerauth.json** file, for example:

```
$ echo "{\"auths\": { \"$REGISTRY\": { \"auth\": $TOKEN } }}" > dockerauth.json
```

7. Do the image mirroring:

```
$ podman run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig \
registry.redhat.io/openshift4/cnf-tests-rhel8:4.15 \
/usr/bin/mirror -registry $REGISTRY/cnftests |  oc image mirror --insecure=true \
-a=$(pwd)/dockerauth.json -f -
```

8. Run the tests:

```
$ podman run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig \
-e LATENCY_TEST_RUNTIME=<time_in_seconds> \
-e IMAGE_REGISTRY=image-registry.openshift-image-registry.svc:5000/cnftests cnf-tests-
local:latest /usr/bin/test-run.sh --ginkgo.v --ginkgo.timeout="24h"
```

Mirroring a different set of test images

You can optionally change the default upstream images that are mirrored for the latency tests.

Procedure

1. The **mirror** command tries to mirror the upstream images by default. This can be overridden by passing a file with the following format to the image:

```
[
    {
        "registry": "public.registry.io:5000",
        "image": "imageforcnftests:4.15"
    }
]
```

2. Pass the file to the **mirror** command, for example saving it locally as **images.json**. With the following command, the local path is mounted in **/kubeconfig** inside the container and that can be passed to the mirror command.

```
$ podman run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig \
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.15 /usr/bin/mirror \
--registry "my.local.registry:5000/" --images "/kubeconfig/images.json" \
|  oc image mirror -f -
```

12.8. TROUBLESHOOTING ERRORS WITH THE CNF-TESTS CONTAINER

To run latency tests, the cluster must be accessible from within the **cnf-tests** container.

Prerequisites

- You have installed the OpenShift CLI (**oc**).

- You have logged in as a user with **cluster-admin** privileges.

Procedure

- Verify that the cluster is accessible from inside the **cnf-tests** container by running the following command:

```
$ podman run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig \
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.15 \
oc get nodes
```

If this command does not work, an error related to spanning across DNS, MTU size, or firewall access might be occurring.

CHAPTER 13. IMPROVING CLUSTER STABILITY IN HIGH LATENCY ENVIRONMENTS USING WORKER LATENCY PROFILES

If the cluster administrator has performed latency tests for platform verification, they can discover the need to adjust the operation of the cluster to ensure stability in cases of high latency. The cluster administrator need change only one parameter, recorded in a file, which controls four parameters affecting how supervisory processes read status and interpret the health of the cluster. Changing only the one parameter provides cluster tuning in an easy, supportable manner.

The **Kubelet** process provides the starting point for monitoring cluster health. The **Kubelet** sets status values for all nodes in the OpenShift Container Platform cluster. The Kubernetes Controller Manager (**kube controller**) reads the status values every 10 seconds, by default. If the **kube controller** cannot read a node status value, it loses contact with that node after a configured period. The default behavior is:

1. The node controller on the control plane updates the node health to **Unhealthy** and marks the node **Ready** condition`Unknown`.

2. In response, the scheduler stops scheduling pods to that node.

3. The Node Lifecycle Controller adds a **node.kubernetes.io/unreachable** taint with a **NoExecute** effect to the node and schedules any pods on the node for eviction after five minutes, by default.

This behavior can cause problems if your network is prone to latency issues, especially if you have nodes at the network edge. In some cases, the Kubernetes Controller Manager might not receive an update from a healthy node due to network latency. The **Kubelet** evicts pods from the node even though the node is healthy.

To avoid this problem, you can use *worker latency profiles* to adjust the frequency that the **Kubelet** and the Kubernetes Controller Manager wait for status updates before taking action. These adjustments help to ensure that your cluster runs properly if network latency between the control plane and the worker nodes is not optimal.

These worker latency profiles contain three sets of parameters that are pre-defined with carefully tuned values to control the reaction of the cluster to increased latency. No need to experimentally find the best values manually.

You can configure worker latency profiles when installing a cluster or at any time you notice increased latency in your cluster network.

13.1. UNDERSTANDING WORKER LATENCY PROFILES

Worker latency profiles are four different categories of carefully-tuned parameters. The four parameters which implement these values are **node-status-update-frequency**, **node-monitor-grace-period**, **default-not-ready-toleration-seconds** and **default-unreachable-toleration-seconds**. These parameters can use values which allow you control the reaction of the cluster to latency issues without needing to determine the best values using manual methods.

> **IMPORTANT**
>
> Setting these parameters manually is not supported. Incorrect parameter settings adversely affect cluster stability.

All worker latency profiles configure the following parameters:

node-status-update-frequency

Specifies how often the kubelet posts node status to the API server.

node-monitor-grace-period

Specifies the amount of time in seconds that the Kubernetes Controller Manager waits for an update from a kubelet before marking the node unhealthy and adding the **node.kubernetes.io/not-ready** or **node.kubernetes.io/unreachable** taint to the node.

default-not-ready-toleration-seconds

Specifies the amount of time in seconds after marking a node unhealthy that the Kube API Server Operator waits before evicting pods from that node.

default-unreachable-toleration-seconds

Specifies the amount of time in seconds after marking a node unreachable that the Kube API Server Operator waits before evicting pods from that node.

The following Operators monitor the changes to the worker latency profiles and respond accordingly:

- The Machine Config Operator (MCO) updates the **node-status-update-frequency** parameter on the worker nodes.

- The Kubernetes Controller Manager updates the **node-monitor-grace-period** parameter on the control plane nodes.

- The Kubernetes API Server Operator updates the **default-not-ready-toleration-seconds** and **default-unreachable-toleration-seconds** parameters on the control plane nodes.

Although the default configuration works in most cases, OpenShift Container Platform offers two other worker latency profiles for situations where the network is experiencing higher latency than usual. The three worker latency profiles are described in the following sections:

Default worker latency profile

With the **Default** profile, each **Kubelet** updates it's status every 10 seconds (**node-status-update-frequency**). The **Kube Controller Manager** checks the statuses of **Kubelet** every 5 seconds (**node-monitor-grace-period**).

The Kubernetes Controller Manager waits 40 seconds for a status update from **Kubelet** before considering the **Kubelet** unhealthy. If no status is made available to the Kubernetes Controller Manager, it then marks the node with the **node.kubernetes.io/not-ready** or **node.kubernetes.io/unreachable** taint and evicts the pods on that node.

If a pod on that node has the **NoExecute** taint, the pod is run according to **tolerationSeconds**. If the pod has no taint, it will be evicted in 300 seconds (**default-not-ready-toleration-seconds** and **default-unreachable-toleration-seconds** settings of the **Kube API Server**).

Profile	Component	Parameter	Value
Default	kubelet	**node-status-update-frequency**	10s
	Kubelet Controller Manager	**node-monitor-grace-period**	40s

Profile	Component	Parameter	Value
	Kubernetes API Server Operator	**default-not-ready-toleration-seconds**	300s
	Kubernetes API Server Operator	**default-unreachable-toleration-seconds**	300s

Medium worker latency profile

Use the **MediumUpdateAverageReaction** profile if the network latency is slightly higher than usual. The **MediumUpdateAverageReaction** profile reduces the frequency of kubelet updates to 20 seconds and changes the period that the Kubernetes Controller Manager waits for those updates to 2 minutes. The pod eviction period for a pod on that node is reduced to 60 seconds. If the pod has the **tolerationSeconds** parameter, the eviction waits for the period specified by that parameter.

The Kubernetes Controller Manager waits for 2 minutes to consider a node unhealthy. In another minute, the eviction process starts.

Profile	Component	Parameter	Value
MediumUpdateAverageReaction	kubelet	**node-status-update-frequency**	20s
	Kubelet Controller Manager	**node-monitor-grace-period**	2m
	Kubernetes API Server Operator	**default-not-ready-toleration-seconds**	60s
	Kubernetes API Server Operator	**default-unreachable-toleration-seconds**	60s

Low worker latency profile

Use the **LowUpdateSlowReaction** profile if the network latency is extremely high. The **LowUpdateSlowReaction** profile reduces the frequency of kubelet updates to 1 minute and changes the period that the Kubernetes Controller Manager waits for those updates to 5 minutes. The pod eviction period for a pod on that node is reduced to 60 seconds. If the pod has the **tolerationSeconds** parameter, the eviction waits for the period specified by that parameter.

The Kubernetes Controller Manager waits for 5 minutes to consider a node unhealthy. In another minute, the eviction process starts.

Profile	Component	Parameter	Value
LowUpdateSlowReaction	kubelet	**node-status-update-frequency**	1m
	Kubelet Controller Manager	**node-monitor-grace-period**	5m
	Kubernetes API Server Operator	**default-not-ready-toleration-seconds**	60s
	Kubernetes API Server Operator	**default-unreachable-toleration-seconds**	60s

13.2. IMPLEMENTING WORKER LATENCY PROFILES AT CLUSTER CREATION

IMPORTANT

To edit the configuration of the installer, you will first need to use the command **openshift-install create manifests** to create the default node manifest as well as other manifest YAML files. This file structure must exist before we can add workerLatencyProfile. The platform on which you are installing may have varying requirements. Refer to the Installing section of the documentation for your specific platform.

The **workerLatencyProfile** must be added to the manifest in the following sequence:

1. Create the manifest needed to build the cluster, using a folder name appropriate for your installation.

2. Create a YAML file to define **config.node**. The file must be in the **manifests** directory.

3. When defining **workerLatencyProfile** in the manifest for the first time, specify any of the profiles at cluster creation time: **Default**, **MediumUpdateAverageReaction** or **LowUpdateSlowReaction**.

Verification

- Here is an example manifest creation showing the **spec.workerLatencyProfile Default** value in the manifest file:

  ```
  $ openshift-install create manifests --dir=<cluster-install-dir>
  ```

- Edit the manifest and add the value. In this example we use **vi** to show an example manifest file with the "Default" **workerLatencyProfile** value added:

```
$ vi <cluster-install-dir>/manifests/config-node-default-profile.yaml
```

Example output

```
apiVersion: config.openshift.io/v1
kind: Node
metadata:
name: cluster
spec:
workerLatencyProfile: "Default"
```

13.3. USING AND CHANGING WORKER LATENCY PROFILES

To change a worker latency profile to deal with network latency, edit the **node.config** object to add the name of the profile. You can change the profile at any time as latency increases or decreases.

You must move one worker latency profile at a time. For example, you cannot move directly from the **Default** profile to the **LowUpdateSlowReaction** worker latency profile. You must move from the **Default** worker latency profile to the **MediumUpdateAverageReaction** profile first, then to **LowUpdateSlowReaction**. Similarly, when returning to the **Default** profile, you must move from the low profile to the medium profile first, then to **Default**.

> **NOTE**
>
> You can also configure worker latency profiles upon installing an OpenShift Container Platform cluster.

Procedure

To move from the default worker latency profile:

1. Move to the medium worker latency profile:

   a. Edit the **node.config** object:

   ```
   $ oc edit nodes.config/cluster
   ```

   b. Add **spec.workerLatencyProfile: MediumUpdateAverageReaction**:

   Example node.config object

   ```
   apiVersion: config.openshift.io/v1
   kind: Node
   metadata:
    annotations:
      include.release.openshift.io/ibm-cloud-managed: "true"
      include.release.openshift.io/self-managed-high-availability: "true"
      include.release.openshift.io/single-node-developer: "true"
      release.openshift.io/create-only: "true"
    creationTimestamp: "2022-07-08T16:02:51Z"
    generation: 1
    name: cluster
    ownerReferences:
   ```

```
  - apiVersion: config.openshift.io/v1
    kind: ClusterVersion
    name: version
    uid: 36282574-bf9f-409e-a6cd-3032939293eb
  resourceVersion: "1865"
  uid: 0c0f7a4c-4307-4187-b591-6155695ac85b
spec:
  workerLatencyProfile: MediumUpdateAverageReaction ❶

# ...
```

❶ Specifies the medium worker latency policy.

Scheduling on each worker node is disabled as the change is being applied.

2. Optional: Move to the low worker latency profile:

a. Edit the **node.config** object:

```
$ oc edit nodes.config/cluster
```

b. Change the **spec.workerLatencyProfile** value to **LowUpdateSlowReaction**:

Example node.config object

```
apiVersion: config.openshift.io/v1
kind: Node
metadata:
  annotations:
    include.release.openshift.io/ibm-cloud-managed: "true"
    include.release.openshift.io/self-managed-high-availability: "true"
    include.release.openshift.io/single-node-developer: "true"
    release.openshift.io/create-only: "true"
  creationTimestamp: "2022-07-08T16:02:51Z"
  generation: 1
  name: cluster
  ownerReferences:
  - apiVersion: config.openshift.io/v1
    kind: ClusterVersion
    name: version
    uid: 36282574-bf9f-409e-a6cd-3032939293eb
  resourceVersion: "1865"
  uid: 0c0f7a4c-4307-4187-b591-6155695ac85b
spec:
  workerLatencyProfile: LowUpdateSlowReaction ❶

# ...
```

❶ Specifies use of the low worker latency policy.

Scheduling on each worker node is disabled as the change is being applied.

Verification

- When all nodes return to the **Ready** condition, you can use the following command to look in the Kubernetes Controller Manager to ensure it was applied:

```
$ oc get KubeControllerManager -o yaml | grep -i workerlatency -A 5 -B 5
```

Example output

```
# ...
    - lastTransitionTime: "2022-07-11T19:47:10Z"
      reason: ProfileUpdated
      status: "False"
      type: WorkerLatencyProfileProgressing
    - lastTransitionTime: "2022-07-11T19:47:10Z"    1
      message: all static pod revision(s) have updated latency profile
      reason: ProfileUpdated
      status: "True"
      type: WorkerLatencyProfileComplete
    - lastTransitionTime: "2022-07-11T19:20:11Z"
      reason: AsExpected
      status: "False"
      type: WorkerLatencyProfileDegraded
    - lastTransitionTime: "2022-07-11T19:20:36Z"
      status: "False"
# ...
```

1 Specifies that the profile is applied and active.

To change the medium profile to default or change the default to medium, edit the **node.config** object and set the **spec.workerLatencyProfile** parameter to the appropriate value.

13.4. EXAMPLE STEPS FOR DISPLAYING RESULTING VALUES OF WORKERLATENCYPROFILE

You can display the values in the **workerLatencyProfile** with the following commands.

Verification

1. Check the **default-not-ready-toleration-seconds** and **default-unreachable-toleration-seconds** fields output by the Kube API Server:

```
$ oc get KubeAPIServer -o yaml | grep -A 1 default-
```

Example output

```
default-not-ready-toleration-seconds:
- "300"
default-unreachable-toleration-seconds:
- "300"
```

2. Check the values of the **node-monitor-grace-period** field from the Kube Controller Manager:

```
$ oc get KubeControllerManager -o yaml | grep -A 1 node-monitor
```

–

Example output

```
node-monitor-grace-period:
- 40s
```

3. Check the **nodeStatusUpdateFrequency** value from the Kubelet. Set the directory **/host** as the root directory within the debug shell. By changing the root directory to **/host**, you can run binaries contained in the host's executable paths:

```
$ oc debug node/<worker-node-name>
$ chroot /host
# cat /etc/kubernetes/kubelet.conf|grep nodeStatusUpdateFrequency
```
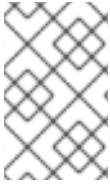
Example output

```
"nodeStatusUpdateFrequency": "10s"
```

These outputs validate the set of timing variables for the Worker Latency Profile.

CHAPTER 14. CREATING A PERFORMANCE PROFILE

Learn about the Performance Profile Creator (PPC) and how you can use it to create a performance profile.

> **NOTE**
>
> Currently, disabling CPU load balancing is not supported by cgroup v2. As a result, you might not get the desired behavior from performance profiles if you have cgroup v2 enabled. Enabling cgroup v2 is not recommended if you are using performance profiles.

14.1. ABOUT THE PERFORMANCE PROFILE CREATOR

The Performance Profile Creator (PPC) is a command-line tool, delivered with the Node Tuning Operator, used to create the performance profile. The tool consumes **must-gather** data from the cluster and several user-supplied profile arguments. The PPC generates a performance profile that is appropriate for your hardware and topology.

The tool is run by one of the following methods:

- Invoking **podman**

- Calling a wrapper script

14.1.1. Gathering data about your cluster using the must-gather command

The Performance Profile Creator (PPC) tool requires **must-gather** data. As a cluster administrator, run the **must-gather** command to capture information about your cluster.

Prerequisites

- Access to the cluster as a user with the **cluster-admin** role.

- The OpenShift CLI (**oc**) installed.

Procedure

1. Optional: Verify that a matching machine config pool exists with a label:

   ```
   $ oc describe mcp/worker-rt
   ```

   Example output

   ```
   Name:         worker-rt
   Namespace:
   Labels:       machineconfiguration.openshift.io/role=worker-rt
   ```

2. If a matching label does not exist add a label for a machine config pool (MCP) that matches with the MCP name:

   ```
   $ oc label mcp <mcp_name> <mcp_name>=""
   ```

3. Navigate to the directory where you want to store the **must-gather** data.

4. Collect cluster information by running the following command:

```
$ oc adm must-gather
```

5. Optional: Create a compressed file from the **must-gather** directory:

```
$ tar cvaf must-gather.tar.gz must-gather/
```

> **NOTE**
>
> Compressed output is required if you are running the Performance Profile Creator wrapper script.

14.1.2. Running the Performance Profile Creator using podman

As a cluster administrator, you can run **podman** and the Performance Profile Creator to create a performance profile.

Prerequisites

- Access to the cluster as a user with the **cluster-admin** role.

- A cluster installed on bare-metal hardware.

- A node with **podman** and OpenShift CLI (**oc**) installed.

- Access to the Node Tuning Operator image.

Procedure

1. Check the machine config pool:

```
$ oc get mcp
```

Example output

```
NAME        CONFIG                                   UPDATED   UPDATING   DEGRADED
MACHINECOUNT   READYMACHINECOUNT   UPDATEDMACHINECOUNT
DEGRADEDMACHINECOUNT   AGE
master      rendered-master-acd1358917e9f98cbdb599aea622d78b     True      False
False    3        3        3           0             22h
worker-cnf   rendered-worker-cnf-1d871ac76e1951d32b2fe92369879826  False     True
False    2        1        1           0             22h
```

2. Use Podman to authenticate to **registry.redhat.io**:

```
$ podman login registry.redhat.io
```

```
Username: <username>
Password: <password>
```

3. Optional: Display help for the PPC tool:

```
$ podman run --rm --entrypoint performance-profile-creator registry.redhat.io/openshift4/ose-
cluster-node-tuning-operator:v4.15 -h
```

Example output

```
A tool that automates creation of Performance Profiles

Usage:
  performance-profile-creator [flags]

Flags:
      --disable-ht                    Disable Hyperthreading
  -h, --help                          help for performance-profile-creator
      --info string                   Show cluster information; requires --must-gather-dir-path,
ignore the other arguments. [Valid values: log, json] (default "log")
      --mcp-name string               MCP name corresponding to the target machines
(required)
      --must-gather-dir-path string   Must gather directory path (default "must-gather")
      --offlined-cpu-count int        Number of offlined CPUs
      --per-pod-power-management      Enable Per Pod Power Management
      --power-consumption-mode string     The power consumption mode.  [Valid values:
default, low-latency, ultra-low-latency] (default "default")
      --profile-name string           Name of the performance profile to be created (default
"performance")
      --reserved-cpu-count int        Number of reserved CPUs (required)
      --rt-kernel                     Enable Real Time Kernel (required)
      --split-reserved-cpus-across-numa   Split the Reserved CPUs across NUMA nodes
      --topology-manager-policy string    Kubelet Topology Manager Policy of the performance
profile to be created. [Valid values: single-numa-node, best-effort, restricted] (default
"restricted")
      --user-level-networking         Run with User level Networking(DPDK) enabled
```

4. Run the Performance Profile Creator tool in discovery mode:

> **NOTE**
>
> Discovery mode inspects your cluster using the output from **must-gather**. The output produced includes information on:
>
> - The NUMA cell partitioning with the allocated CPU ids
>
> - Whether hyperthreading is enabled
>
> Using this information you can set appropriate values for some of the arguments supplied to the Performance Profile Creator tool.

```
$ podman run --entrypoint performance-profile-creator -v <path_to_must-gather>/must-
gather:/must-gather:z registry.redhat.io/openshift4/ose-cluster-node-tuning-operator:v4.15 --
info log --must-gather-dir-path /must-gather
```

> **NOTE**
>
> This command uses the performance profile creator as a new entry point to
> **podman**. It maps the **must-gather** data for the host into the container image
> and invokes the required user-supplied profile arguments to produce the **my-performance-profile.yaml** file.
>
> The **-v** option can be the path to either:
>
> - The **must-gather** output directory
>
> - An existing directory containing the **must-gather** decompressed tarball
>
> The **info** option requires a value which specifies the output format. Possible
> values are log and JSON. The JSON format is reserved for debugging.

5. Run **podman**:

```
$ podman run --entrypoint performance-profile-creator -v /must-gather:/must-gather:z
registry.redhat.io/openshift4/ose-cluster-node-tuning-operator:v4.15 --mcp-name=worker-cnf
--reserved-cpu-count=4 --rt-kernel=true --split-reserved-cpus-across-numa=false --must-
gather-dir-path /must-gather --power-consumption-mode=ultra-low-latency --offlined-cpu-
count=6 > my-performance-profile.yaml
```

> **NOTE**
>
> The Performance Profile Creator arguments are shown in the Performance
> Profile Creator arguments table. The following arguments are required:
>
> - **reserved-cpu-count**
>
> - **mcp-name**
>
> - **rt-kernel**
>
> The **mcp-name** argument in this example is set to **worker-cnf** based on the
> output of the command **oc get mcp**. For single-node OpenShift use **--mcp-name=master**.

6. Review the created YAML file:

```
$ cat my-performance-profile.yaml
```

Example output

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: performance
spec:
  cpu:
    isolated: 2-39,48-79
    offlined: 42-47
    reserved: 0-1,40-41
```

```
      machineConfigPoolSelector:
        machineconfiguration.openshift.io/role: worker-cnf
      nodeSelector:
        node-role.kubernetes.io/worker-cnf: ""
      numa:
        topologyPolicy: restricted
      realTimeKernel:
        enabled: true
      workloadHints:
        highPowerConsumption: true
        realTime: true
```

7. Apply the generated profile:

```
$ oc apply -f my-performance-profile.yaml
```

14.1.2.1. How to run **podman** to create a performance profile

The following example illustrates how to run **podman** to create a performance profile with 20 reserved CPUs that are to be split across the NUMA nodes.

Node hardware configuration:

- 80 CPUs

- Hyperthreading enabled

- Two NUMA nodes

- Even numbered CPUs run on NUMA node 0 and odd numbered CPUs run on NUMA node 1

Run **podman** to create the performance profile:

```
$ podman run --entrypoint performance-profile-creator -v /must-gather:/must-gather:z
registry.redhat.io/openshift4/ose-cluster-node-tuning-operator:v4.15 --mcp-name=worker-cnf --
reserved-cpu-count=20 --rt-kernel=true --split-reserved-cpus-across-numa=true --must-gather-dir-
path /must-gather > my-performance-profile.yaml
```

The created profile is described in the following YAML:

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: performance
spec:
  cpu:
    isolated: 10-39,50-79
    reserved: 0-9,40-49
  nodeSelector:
    node-role.kubernetes.io/worker-cnf: ""
  numa:
    topologyPolicy: restricted
  realTimeKernel:
    enabled: true
```

> **NOTE**
>
> In this case, 10 CPUs are reserved on NUMA node 0 and 10 are reserved on NUMA node 1.

14.1.3. Running the Performance Profile Creator wrapper script

The performance profile wrapper script simplifies the running of the Performance Profile Creator (PPC) tool. It hides the complexities associated with running **podman** and specifying the mapping directories and it enables the creation of the performance profile.

Prerequisites

- Access to the Node Tuning Operator image.

- Access to the **must-gather** tarball.

Procedure

1. Create a file on your local machine named, for example, **run-perf-profile-creator.sh**:

   ```
   $ vi run-perf-profile-creator.sh
   ```

2. Paste the following code into the file:

   ```bash
   #!/bin/bash

   readonly CONTAINER_RUNTIME=${CONTAINER_RUNTIME:-podman}
   readonly CURRENT_SCRIPT=$(basename "$0")
   readonly CMD="${CONTAINER_RUNTIME} run --entrypoint performance-profile-creator"
   readonly IMG_EXISTS_CMD="${CONTAINER_RUNTIME} image exists"
   readonly IMG_PULL_CMD="${CONTAINER_RUNTIME} image pull"
   readonly MUST_GATHER_VOL="/must-gather"

   NTO_IMG="registry.redhat.io/openshift4/ose-cluster-node-tuning-operator:v4.15"
   MG_TARBALL=""
   DATA_DIR=""

   usage() {
     print "Wrapper usage:"
     print "  ${CURRENT_SCRIPT} [-h] [-p image][-t path] -- [performance-profile-creator flags]"
     print ""
     print "Options:"
     print "   -h                 help for ${CURRENT_SCRIPT}"
     print "   -p                 Node Tuning Operator image"
     print "   -t                 path to a must-gather tarball"

     ${IMG_EXISTS_CMD} "${NTO_IMG}" && ${CMD} "${NTO_IMG}" -h
   }

   function cleanup {
     [ -d "${DATA_DIR}" ] && rm -rf "${DATA_DIR}"
   }
   trap cleanup EXIT

   exit_error() {
   ```

```
  print "error: $*"
  usage
  exit 1
}

print() {
  echo  "$*" >&2
}

check_requirements() {
  ${IMG_EXISTS_CMD} "${NTO_IMG}" || ${IMG_PULL_CMD} "${NTO_IMG}" || \
     exit_error "Node Tuning Operator image not found"

  [ -n "${MG_TARBALL}" ] || exit_error "Must-gather tarball file path is mandatory"
  [ -f "${MG_TARBALL}" ] || exit_error "Must-gather tarball file not found"

  DATA_DIR=$(mktemp -d -t "${CURRENT_SCRIPT}XXXX") || exit_error "Cannot create the
data directory"
  tar -zxf "${MG_TARBALL}" --directory "${DATA_DIR}" || exit_error "Cannot decompress the
must-gather tarball"
  chmod a+rx "${DATA_DIR}"

  return 0
}

main() {
  while getopts ':hp:t:' OPT; do
   case "${OPT}" in
     h)
       usage
       exit 0
       ;;
     p)
       NTO_IMG="${OPTARG}"
       ;;
     t)
       MG_TARBALL="${OPTARG}"
       ;;
     ?)
       exit_error "invalid argument: ${OPTARG}"
       ;;
   esac
  done
  shift $((OPTIND - 1))

  check_requirements || exit 1

  ${CMD} -v "${DATA_DIR}:${MUST_GATHER_VOL}:z" "${NTO_IMG}" "$@" --must-gather-
dir-path "${MUST_GATHER_VOL}"
  echo "" 1>&2
}

main "$@"
```

3. Add execute permissions for everyone on this script:

```
$ chmod a+x run-perf-profile-creator.sh
```

4. Optional: Display the **run-perf-profile-creator.sh** command usage:

```
$ ./run-perf-profile-creator.sh -h
```

Expected output

```
Wrapper usage:
  run-perf-profile-creator.sh [-h] [-p image][-t path] -- [performance-profile-creator flags]

Options:
  -h             help for run-perf-profile-creator.sh
  -p             Node Tuning Operator image ❶
  -t             path to a must-gather tarball ❷
A tool that automates creation of Performance Profiles

Usage:
  performance-profile-creator [flags]

Flags:
      --disable-ht                   Disable Hyperthreading
  -h, --help                         help for performance-profile-creator
      --info string                  Show cluster information; requires --must-gather-dir-path,
ignore the other arguments. [Valid values: log, json] (default "log")
      --mcp-name string              MCP name corresponding to the target machines
(required)
      --must-gather-dir-path string       Must gather directory path (default "must-gather")
      --offlined-cpu-count int           Number of offlined CPUs
      --per-pod-power-management          Enable Per Pod Power Management
      --power-consumption-mode string      The power consumption mode.  [Valid values:
default, low-latency, ultra-low-latency] (default "default")
      --profile-name string              Name of the performance profile to be created (default
"performance")
      --reserved-cpu-count int           Number of reserved CPUs (required)
      --rt-kernel                    Enable Real Time Kernel (required)
      --split-reserved-cpus-across-numa    Split the Reserved CPUs across NUMA nodes
      --topology-manager-policy string    Kubelet Topology Manager Policy of the performance
profile to be created. [Valid values: single-numa-node, best-effort, restricted] (default
"restricted")
      --user-level-networking            Run with User level Networking(DPDK) enabled
```

> **NOTE**
>
> There two types of arguments:
>
> - Wrapper arguments namely **-h**, **-p** and **-t**
>
> - PPC arguments

❶ Optional: Specify the Node Tuning Operator image. If not set, the default upstream image is used: **registry.redhat.io/openshift4/ose-cluster-node-tuning-operator:v4.15**.

2 **-t** is a required wrapper script argument and specifies the path to a **must-gather** tarball.

5. Run the performance profile creator tool in discovery mode:

> **NOTE**
>
> Discovery mode inspects your cluster using the output from **must-gather**. The output produced includes information on:
>
> - The NUMA cell partitioning with the allocated CPU IDs
>
> - Whether hyperthreading is enabled
>
> Using this information you can set appropriate values for some of the arguments supplied to the Performance Profile Creator tool.

```
$ ./run-perf-profile-creator.sh -t /must-gather/must-gather.tar.gz -- --info=log
```

> **NOTE**
>
> The **info** option requires a value which specifies the output format. Possible values are log and JSON. The JSON format is reserved for debugging.

6. Check the machine config pool:

```
$ oc get mcp
```

Example output

```
NAME        CONFIG                                       UPDATED   UPDATING   DEGRADED
MACHINECOUNT   READYMACHINECOUNT   UPDATEDMACHINECOUNT
DEGRADEDMACHINECOUNT   AGE
master      rendered-master-acd1358917e9f98cbdb599aea622d78b      True      False
False     3         3             3                 0                   22h
worker-cnf  rendered-worker-cnf-1d871ac76e1951d32b2fe92369879826  False     True
False     2         1             1                 0                   22h
```

7. Create a performance profile:

```
$ ./run-perf-profile-creator.sh -t /must-gather/must-gather.tar.gz -- --mcp-name=worker-cnf --reserved-cpu-count=2 --rt-kernel=true > my-performance-profile.yaml
```

> **NOTE**
>
> The Performance Profile Creator arguments are shown in the Performance Profile Creator arguments table. The following arguments are required:
>
> - **reserved-cpu-count**
>
> - **mcp-name**
>
> - **rt-kernel**
>
> The **mcp-name** argument in this example is set to **worker-cnf** based on the output of the command **oc get mcp**. For single-node OpenShift use **--mcp-name=master**.

8. Review the created YAML file:

   ```
   $ cat my-performance-profile.yaml
   ```

   Example output

   ```
   apiVersion: performance.openshift.io/v2
   kind: PerformanceProfile
   metadata:
     name: performance
   spec:
     cpu:
       isolated: 1-39,41-79
       reserved: 0,40
     nodeSelector:
       node-role.kubernetes.io/worker-cnf: ""
     numa:
       topologyPolicy: restricted
     realTimeKernel:
       enabled: false
   ```

9. Apply the generated profile:

   > **NOTE**
   >
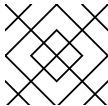   > Install the Node Tuning Operator before applying the profile.

   ```
   $ oc apply -f my-performance-profile.yaml
   ```

14.1.4. Performance Profile Creator arguments

Table 14.1. Performance Profile Creator arguments

Argument	Description

Argument	Description
disable-ht	Disable hyperthreading. Possible values: **true** or **false**. Default: **false**. ⚠️ **WARNING** If this argument is set to **true** you should not disable hyperthreading in the BIOS. Disabling hyperthreading is accomplished with a kernel command line argument.
info	This captures cluster information and is used in discovery mode only. Discovery mode also requires the **must-gather-dir-path** argument. If any other arguments are set they are ignored. Possible values: ● **log** ● **JSON** **NOTE** These options define the output format with the JSON format being reserved for debugging. Default: **log**.
mcp-name	MCP name for example **worker-cnf** corresponding to the target machines. This parameter is required.
must-gather-dir-path	Must gather directory path. This parameter is required. When the user runs the tool with the wrapper script **must-gather** is supplied by the script itself and the user must not specify it.

Argument	Description
offlined-cpu-count	Number of offlined CPUs. **NOTE** This must be a natural number greater than 0. If not enough logical processors are offlined then error messages are logged. The messages are: > Error: failed to compute the reserved and isolated CPUs: please ensure that reserved-cpu-count plus offlined-cpu-count should be in the range [0,1] > Error: failed to compute the reserved and isolated CPUs: please specify the offlined CPU count in the range [0,1]
power-consumption-mode	The power consumption mode. Possible values: - **default**: CPU partitioning with enabled power management and basic low-latency. - **low-latency**: Enhanced measures to improve latency figures. - **ultra-low-latency**: Priority given to optimal latency, at the expense of power management. Default: **default**.
per-pod-power-management	Enable per pod power management. You cannot use this argument if you configured **ultra-low-latency** as the power consumption mode. Possible values: **true** or **false**. Default: **false**.
profile-name	Name of the performance profile to create. Default: **performance**.
reserved-cpu-count	Number of reserved CPUs. This parameter is required. **NOTE** This must be a natural number. A value of 0 is not allowed.
rt-kernel	Enable real-time kernel. This parameter is required. Possible values: **true** or **false**.

Argument	Description
split-reserved-cpus-across-numa	Split the reserved CPUs across NUMA nodes. Possible values: **true** or **false**. Default: **false**.
topology-manager-policy	Kubelet Topology Manager policy of the performance profile to be created. Possible values: • **single-numa-node** • **best-effort** • **restricted** Default: **restricted**.
user-level-networking	Run with user level networking (DPDK) enabled. Possible values: **true** or **false**. Default: **false**.

14.2. REFERENCE PERFORMANCE PROFILES

14.2.1. A performance profile template for clusters that use OVS-DPDK on OpenStack

To maximize machine performance in a cluster that uses Open vSwitch with the Data Plane Development Kit (OVS-DPDK) on Red Hat OpenStack Platform (RHOSP), you can use a performance profile.

You can use the following performance profile template to create a profile for your deployment.

A performance profile template for clusters that use OVS-DPDK

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: cnf-performanceprofile
spec:
  additionalKernelArgs:
    - nmi_watchdog=0
    - audit=0
    - mce=off
    - processor.max_cstate=1
    - idle=poll
    - intel_idle.max_cstate=0
    - default_hugepagesz=1GB
    - hugepagesz=1G
```

```
    - intel_iommu=on
cpu:
  isolated: <CPU_ISOLATED>
  reserved: <CPU_RESERVED>
hugepages:
  defaultHugepagesSize: 1G
  pages:
    - count: <HUGEPAGES_COUNT>
      node: 0
      size: 1G
nodeSelector:
  node-role.kubernetes.io/worker: ''
realTimeKernel:
  enabled: false
  globallyDisableIrqLoadBalancing: true
```

Insert values that are appropriate for your configuration for the **CPU_ISOLATED**, **CPU_RESERVED**, and **HUGEPAGES_COUNT** keys.

To learn how to create and use performance profiles, see the "Creating a performance profile" page in the "Scalability and performance" section of the OpenShift Container Platform documentation.

14.3. ADDITIONAL RESOURCES

- For more information about the **must-gather** tool, see Gathering data about your cluster .

CHAPTER 15. WORKLOAD PARTITIONING

In resource-constrained environments, you can use workload partitioning to isolate OpenShift Container Platform services, cluster management workloads, and infrastructure pods to run on a reserved set of CPUs.

The minimum number of reserved CPUs required for the cluster management is four CPU Hyper-Threads (HTs). With workload partitioning, you annotate the set of cluster management pods and a set of typical add-on Operators for inclusion in the cluster management workload partition. These pods operate normally within the minimum size CPU configuration. Additional Operators or workloads outside of the set of minimum cluster management pods require additional CPUs to be added to the workload partition.

Workload partitioning isolates user workloads from platform workloads using standard Kubernetes scheduling capabilities.

The following changes are required for workload partitioning:

1. In the **install-config.yaml** file, add the additional field: **cpuPartitioningMode**.

   ```
   apiVersion: v1
   baseDomain: devcluster.openshift.com
   cpuPartitioningMode: AllNodes ❶
   compute:
     - architecture: amd64
       hyperthreading: Enabled
       name: worker
       platform: {}
       replicas: 3
   controlPlane:
     architecture: amd64
     hyperthreading: Enabled
     name: master
     platform: {}
     replicas: 3
   ```

   ❶ Sets up a cluster for CPU partitioning at install time. The default value is **None**.

   > **NOTE**
   >
   > Workload partitioning can only be enabled during cluster installation. You cannot disable workload partitioning postinstallation.

2. In the performance profile, specify the **isolated** and **reserved** CPUs.

   Recommended performance profile configuration

   ```
   apiVersion: performance.openshift.io/v2
   kind: PerformanceProfile
   metadata:
     # if you change this name make sure the 'include' line in TunedPerformancePatch.yaml
     # matches this name: include=openshift-node-performance-
   ${PerformanceProfile.metadata.name}
     # Also in file 'validatorCRs/informDuValidator.yaml':
   ```

```
# name: 50-performance-${PerformanceProfile.metadata.name}
name: openshift-node-performance-profile
annotations:
  ran.openshift.io/reference-configuration: "ran-du.redhat.com"
spec:
  additionalKernelArgs:
    - "rcupdate.rcu_normal_after_boot=0"
    - "efi=runtime"
    - "vfio_pci.enable_sriov=1"
    - "vfio_pci.disable_idle_d3=1"
    - "module_blacklist=irdma"
  cpu:
    isolated: $isolated
    reserved: $reserved
  hugepages:
    defaultHugepagesSize: $defaultHugepagesSize
    pages:
      - size: $size
        count: $count
        node: $node
  machineConfigPoolSelector:
    pools.operator.machineconfiguration.openshift.io/$mcp: ""
  nodeSelector:
    node-role.kubernetes.io/$mcp: ''
  numa:
    topologyPolicy: "restricted"
  # To use the standard (non-realtime) kernel, set enabled to false
  realTimeKernel:
    enabled: true
  workloadHints:
    # WorkloadHints defines the set of upper level flags for different type of workloads.
    # See https://github.com/openshift/cluster-node-tuning-
operator/blob/master/docs/performanceprofile/performance_profile.md#workloadhints
    # for detailed descriptions of each item.
    # The configuration below is set for a low latency, performance mode.
    realTime: true
    highPowerConsumption: false
    perPodPowerManagement: false
```

Table 15.1. PerformanceProfile CR options for single-node OpenShift clusters

PerformanceProfile CR field	Description

PerformanceProfile CR field	Description
metadata.name	Ensure that **name** matches the following fields set in related GitOps ZTP custom resources (CRs): • **include=openshift-node-performance-${PerformanceProfile.metadata.name}** in **TunedPerformancePatch.yaml** • **name: 50-performance-${PerformanceProfile.metadata.name}** in **validatorCRs/informDuValidator.yaml**
spec.additionalKernelArgs	**"efi=runtime"** Configures UEFI secure boot for the cluster host.
spec.cpu.isolated	Set the isolated CPUs. Ensure all of the Hyper-Threading pairs match. IMPORTANT The reserved and isolated CPU pools must not overlap and together must span all available cores. CPU cores that are not accounted for cause an undefined behaviour in the system.
spec.cpu.reserved	Set the reserved CPUs. When workload partitioning is enabled, system processes, kernel threads, and system container threads are restricted to these CPUs. All CPUs that are not isolated should be reserved.
spec.hugepages.pages	• Set the number of huge pages (**count**) • Set the huge pages size (**size**). • Set **node** to the NUMA node where the **hugepages** are allocated (**node**)
spec.realTimeKernel	Set **enabled** to **true** to use the realtime kernel.
spec.workloadHints	Use **workloadHints** to define the set of top level flags for different type of workloads. The example configuration configures the cluster for low latency and high performance.

Workload partitioning introduces an extended **management.workload.openshift.io/cores** resource type for platform pods. kubelet advertises the resources and CPU requests by pods allocated to the pool within the corresponding resource. When workload partitioning is enabled, the **management.workload.openshift.io/cores** resource allows the scheduler to correctly assign pods based on the **cpushares** capacity of the host, not just the default **cpuset**.

Additional resources

- For the recommended workload partitioning configuration for single-node OpenShift clusters, see Workload partitioning.

CHAPTER 16. USING THE NODE OBSERVABILITY OPERATOR

The Node Observability Operator collects and stores CRI-O and Kubelet profiling or metrics from scripts of compute nodes.

With the Node Observability Operator, you can query the profiling data, enabling analysis of performance trends in CRI-O and Kubelet. It supports debugging performance-related issues and executing embedded scripts for network metrics by using the **run** field in the custom resource definition. To enable CRI-O and Kubelet profiling or scripting, you can configure the **type** field in the custom resource definition.

> **IMPORTANT**
>
> The Node Observability Operator is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.
>
> For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

16.1. WORKFLOW OF THE NODE OBSERVABILITY OPERATOR

The following workflow outlines on how to query the profiling data using the Node Observability Operator:

1. Install the Node Observability Operator in the OpenShift Container Platform cluster.

2. Create a NodeObservability custom resource to enable the CRI-O profiling on the worker nodes of your choice.

3. Run the profiling query to generate the profiling data.

16.2. INSTALLING THE NODE OBSERVABILITY OPERATOR

The Node Observability Operator is not installed in OpenShift Container Platform by default. You can install the Node Observability Operator by using the OpenShift Container Platform CLI or the web console.

16.2.1. Installing the Node Observability Operator using the CLI

You can install the Node Observability Operator by using the OpenShift CLI (oc).

Prerequisites

- You have installed the OpenShift CLI (oc).

- You have access to the cluster with **cluster-admin** privileges.

Procedure

1. Confirm that the Node Observability Operator is available by running the following command:

```
$ oc get packagemanifests -n openshift-marketplace node-observability-operator
```

Example output

```
NAME                    CATALOG            AGE
node-observability-operator     Red Hat Operators      9h
```

2. Create the **node-observability-operator** namespace by running the following command:

```
$ oc new-project node-observability-operator
```

3. Create an **OperatorGroup** object YAML file:

```
cat <<EOF | oc apply -f -
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: node-observability-operator
  namespace: node-observability-operator
spec:
  targetNamespaces: []
EOF
```

4. Create a **Subscription** object YAML file to subscribe a namespace to an Operator:

```
cat <<EOF | oc apply -f -
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: node-observability-operator
  namespace: node-observability-operator
spec:
  channel: alpha
  name: node-observability-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
EOF
```

Verification

1. View the install plan name by running the following command:

```
$ oc -n node-observability-operator get sub node-observability-operator -o yaml | yq
'.status.installplan.name'
```

Example output

```
install-dt54w
```

2. Verify the install plan status by running the following command:

```
$ oc -n node-observability-operator get ip <install_plan_name> -o yaml | yq '.status.phase'
```

<install_plan_name> is the install plan name that you obtained from the output of the previous command.

Example output

```
COMPLETE
```

3. Verify that the Node Observability Operator is up and running:

```
$ oc get deploy -n node-observability-operator
```

Example output

```
NAME                                        READY   UP-TO-DATE   AVAILABLE   AGE
node-observability-operator-controller-manager  1/1     1            1           40h
```

16.2.2. Installing the Node Observability Operator using the web console

You can install the Node Observability Operator from the OpenShift Container Platform web console.

Prerequisites

- You have access to the cluster with **cluster-admin** privileges.

- You have access to the OpenShift Container Platform web console.

Procedure

1. Log in to the OpenShift Container Platform web console.

2. In the Administrator's navigation panel, expand **Operators → OperatorHub**.

3. In the **All items** field, enter **Node Observability Operator** and select the **Node Observability Operator** tile.

4. Click **Install**.

5. On the **Install Operator** page, configure the following settings:

   a. In the **Update channel** area, click **alpha**.

   b. In the **Installation mode** area, click **A specific namespace on the cluster**.

   c. From the **Installed Namespace** list, select **node-observability-operator** from the list.

   d. In the **Update approval** area, select **Automatic**.

   e. Click **Install**.

Verification

1. In the Administrator's navigation panel, expand **Operators → Installed Operators**.

2. Verify that the Node Observability Operator is listed in the Operators list.

16.3. REQUESTING CRI-O AND KUBELET PROFILING DATA USING THE NODE OBSERVABILITY OPERATOR

Creating a Node Observability custom resource to collect CRI-O and Kubelet profiling data.

16.3.1. Creating the Node Observability custom resource

You must create and run the **NodeObservability** custom resource (CR) before you run the profiling query. When you run the **NodeObservability** CR, it creates the necessary machine config and machine config pool CRs to enable the CRI-O profiling on the worker nodes matching the **nodeSelector**.

> **IMPORTANT**
>
> If CRI-O profiling is not enabled on the worker nodes, the **NodeObservabilityMachineConfig** resource gets created. Worker nodes matching the **nodeSelector** specified in **NodeObservability** CR restarts. This might take 10 or more minutes to complete.

> **NOTE**
>
> Kubelet profiling is enabled by default.

The CRI-O unix socket of the node is mounted on the agent pod, which allows the agent to communicate with CRI-O to run the pprof request. Similarly, the **kubelet-serving-ca** certificate chain is mounted on the agent pod, which allows secure communication between the agent and node's kubelet endpoint.

Prerequisites

- You have installed the Node Observability Operator.

- You have installed the OpenShift CLI (oc).

- You have access to the cluster with **cluster-admin** privileges.

Procedure

1. Log in to the OpenShift Container Platform CLI by running the following command:

   ```
   $ oc login -u kubeadmin https://<HOSTNAME>:6443
   ```

2. Switch back to the **node-observability-operator** namespace by running the following command:

   ```
   $ oc project node-observability-operator
   ```

3. Create a CR file named **nodeobservability.yaml** that contains the following text:

   ```
   apiVersion: nodeobservability.olm.openshift.io/v1alpha2
   kind: NodeObservability
   metadata:
    name: cluster 1
   spec:
   ```

```
    nodeSelector:
      kubernetes.io/hostname: <node_hostname> 2
    type: crio-kubelet
```

1 You must specify the name as **cluster** because there should be only one **NodeObservability** CR per cluster.

2 Specify the nodes on which the Node Observability agent must be deployed.

4. Run the **NodeObservability** CR:

```
oc apply -f nodeobservability.yaml
```

Example output

```
nodeobservability.olm.openshift.io/cluster created
```

5. Review the status of the **NodeObservability** CR by running the following command:

```
$ oc get nob/cluster -o yaml | yq '.status.conditions'
```

Example output

```
conditions:
  conditions:
  - lastTransitionTime: "2022-07-05T07:33:54Z"
    message: 'DaemonSet node-observability-ds ready: true NodeObservabilityMachineConfig
      ready: true'
    reason: Ready
    status: "True"
    type: Ready
```

NodeObservability CR run is completed when the reason is **Ready** and the status is **True**.

16.3.2. Running the profiling query

To run the profiling query, you must create a **NodeObservabilityRun** resource. The profiling query is a blocking operation that fetches CRI-O and Kubelet profiling data for a duration of 30 seconds. After the profiling query is complete, you must retrieve the profiling data inside the container file system **/run/node-observability** directory. The lifetime of data is bound to the agent pod through the **emptyDir** volume, so you can access the profiling data while the agent pod is in the **running** status.
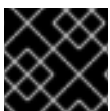
> **IMPORTANT**
>
> You can request only one profiling query at any point of time.

Prerequisites

- You have installed the Node Observability Operator.

- You have created the **NodeObservability** custom resource (CR).

- You have access to the cluster with **cluster-admin** privileges.

Procedure

1. Create a **NodeObservabilityRun** resource file named **nodeobservabilityrun.yaml** that contains the following text:

```
apiVersion: nodeobservability.olm.openshift.io/v1alpha2
kind: NodeObservabilityRun
metadata:
  name: nodeobservabilityrun
spec:
  nodeObservabilityRef:
    name: cluster
```

2. Trigger the profiling query by running the **NodeObservabilityRun** resource:

```
$ oc apply -f nodeobservabilityrun.yaml
```

3. Review the status of the **NodeObservabilityRun** by running the following command:

```
$ oc get nodeobservabilityrun nodeobservabilityrun -o yaml  | yq '.status.conditions'
```

Example output

```
conditions:
- lastTransitionTime: "2022-07-07T14:57:34Z"
  message: Ready to start profiling
  reason: Ready
  status: "True"
  type: Ready
- lastTransitionTime: "2022-07-07T14:58:10Z"
  message: Profiling query done
  reason: Finished
  status: "True"
  type: Finished
```

The profiling query is complete once the status is **True** and type is **Finished**.

4. Retrieve the profiling data from the container's **/run/node-observability** path by running the following bash script:

```
for a in $(oc get nodeobservabilityrun nodeobservabilityrun -o yaml | yq
.status.agents[].name); do
  echo "agent ${a}"
  mkdir -p "/tmp/${a}"
  for p in $(oc exec "${a}" -c node-observability-agent -- bash -c "ls /run/node-
observability/*.pprof"); do
    f="$(basename ${p})"
    echo "copying ${f} to /tmp/${a}/${f}"
    oc exec "${a}" -c node-observability-agent -- cat "${p}" > "/tmp/${a}/${f}"
  done
done
```

16.4. NODE OBSERVABILITY OPERATOR SCRIPTING

Scripting allows you to run pre-configured bash scripts, using the current Node Observability Operator and Node Observability Agent.

These scripts monitor key metrics like CPU load, memory pressure, and worker node issues. They also collect sar reports and custom performance metrics.

16.4.1. Creating the Node Observability custom resource for scripting

You must create and run the **NodeObservability** custom resource (CR) before you run the scripting. When you run the **NodeObservability** CR, it enables the agent in scripting mode on the compute nodes matching the **nodeSelector** label.

Prerequisites

- You have installed the Node Observability Operator.

- You have installed the OpenShift CLI (**oc**).

- You have access to the cluster with **cluster-admin** privileges.

Procedure

1. Log in to the OpenShift Container Platform cluster by running the following command:

   ```
   $ oc login -u kubeadmin https://<host_name>:6443
   ```

2. Switch to the **node-observability-operator** namespace by running the following command:

   ```
   $ oc project node-observability-operator
   ```

3. Create a file named **nodeobservability.yaml** that contains the following content:

   ```
   apiVersion: nodeobservability.olm.openshift.io/v1alpha2
   kind: NodeObservability
   metadata:
     name: cluster 1
   spec:
     nodeSelector:
       kubernetes.io/hostname: <node_hostname> 2
     type: scripting 3
   ```

   1 You must specify the name as **cluster** because there should be only one **NodeObservability** CR per cluster.

   2 Specify the nodes on which the Node Observability agent must be deployed.

   3 To deploy the agent in scripting mode, you must set the type to **scripting**.

4. Create the **NodeObservability** CR by running the following command:

   ```
   $ oc apply -f nodeobservability.yaml
   ```

Example output

```
nodeobservability.olm.openshift.io/cluster created
```

5. Review the status of the **NodeObservability** CR by running the following command:

```
$ oc get nob/cluster -o yaml | yq '.status.conditions'
```

Example output

```
conditions:
  conditions:
  - lastTransitionTime: "2022-07-05T07:33:54Z"
    message: 'DaemonSet node-observability-ds ready: true NodeObservabilityScripting
      ready: true'
    reason: Ready
    status: "True"
    type: Ready
```

The **NodeObservability** CR run is completed when the **reason** is **Ready** and **status** is **"True"**.

16.4.2. Configuring Node Observability Operator scripting

Prerequisites

- You have installed the Node Observability Operator.

- You have created the **NodeObservability** custom resource (CR).

- You have access to the cluster with **cluster-admin** privileges.

Procedure

1. Create a file named **nodeobservabilityrun-script.yaml** that contains the following content:

```
apiVersion: nodeobservability.olm.openshift.io/v1alpha2
kind: NodeObservabilityRun
metadata:
  name: nodeobservabilityrun-script
  namespace: node-observability-operator
spec:
  nodeObservabilityRef:
    name: cluster
    type: scripting
```

> **IMPORTANT**
>
> You can request only the following scripts:
>
> - **metrics.sh**
>
> - **network-metrics.sh** (uses **monitor.sh**)

2. Trigger the scripting by creating the **NodeObservabilityRun** resource with the following command:

```
$ oc apply -f nodeobservabilityrun-script.yaml
```

3. Review the status of the **NodeObservabilityRun** scripting by running the following command:

```
$ oc get nodeobservabilityrun nodeobservabilityrun-script -o yaml  | yq '.status.conditions'
```

Example output

```
Status:
  Agents:
    Ip:    10.128.2.252
    Name:  node-observability-agent-n2fpm
    Port:  8443
    Ip:    10.131.0.186
    Name:  node-observability-agent-wcc8p
    Port:  8443
  Conditions:
    Conditions:
      Last Transition Time:  2023-12-19T15:10:51Z
      Message:             Ready to start profiling
      Reason:             Ready
      Status:             True
      Type:               Ready
      Last Transition Time:  2023-12-19T15:11:01Z
      Message:             Profiling query done
      Reason:             Finished
      Status:             True
      Type:               Finished
  Finished Timestamp:     2023-12-19T15:11:01Z
  Start Timestamp:        2023-12-19T15:10:51Z
```

The scripting is complete once **Status** is **True** and **Type** is **Finished**.

4. Retrieve the scripting data from the root path of the container by running the following bash script:

```bash
#!/bin/bash

RUN=$(oc get nodeobservabilityrun --no-headers | awk '{print $1}')

for a in $(oc get nodeobservabilityruns.nodeobservability.olm.openshift.io/${RUN} -o json | jq .status.agents[].name); do
  echo "agent ${a}"
  agent=$(echo ${a} | tr -d "\"\'\`")
  base_dir=$(oc exec "${agent}" -c node-observability-agent -- bash -c "ls -t | grep node-observability-agent" | head -1)
  echo "${base_dir}"
  mkdir -p "/tmp/${agent}"
  for p in $(oc exec "${agent}" -c node-observability-agent -- bash -c "ls ${base_dir}"); do
    f="/${base_dir}/${p}"
    echo "copying ${f} to /tmp/${agent}/${p}"
```

```
    oc exec "${agent}" -c node-observability-agent -- cat ${f} > "/tmp/${agent}/${p}"
  done
done
```

16.5. ADDITIONAL RESOURCES

For more information on how to collect worker metrics, see Red Hat Knowledgebase article .