Red Hat build of MicroShift 4.17

Configuring

Configuring MicroShift

Last Updated: 2024-12-06

Red Hat build of MicroShift 4.17 Configuring

Configuring MicroShift

Legal Notice

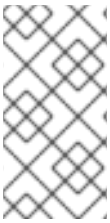
Abstract

This document provides instructions for configuring MicroShift.

Table of Contents

CHAPTER 1. USING THE MICROSHIFT CONFIGURATION FILE

A YAML file customizes MicroShift instances with your preferences, settings, and parameters.



NOTE

If you want to make configuration changes or deploy applications through the MicroShift API with tools other than **kustomize** manifests, you must wait until the greenboot health checks have finished. This ensures that your changes are not lost if greenboot rolls your **rpm-ostree** system back to an earlier state.

1.1. THE MICROSHIFT YAML CONFIGURATION FILE

At start up, MicroShift checks the system-wide **/etc/microshift/** directory for a configuration file named **config.yaml**. If the configuration file does not exist in the directory, built-in default values are used to start the service.

The MicroShift configuration file must be used in combination with host and, sometimes, application and service settings. Ensure that each item is configured in tandem when you customize your MicroShift cluster.

1.1.1. Default settings

If you do not create a **config.yaml** file, default values are used. The following example shows the default configuration settings.

- To see the default values, run the following command:

```
$ microshift show-config
```

Default values example output in YAML form

```
apiServer:
  advertiseAddress: 10.44.0.0/32 1
  auditLog:
    maxFileAge: 0
    maxFileSize: 200
    maxFiles: 10
    profile: Default
  namedCertificates:
    - certPath: ""
      keyPath: ""
      names:
        - ""
  subjectAltNames: []
debugging:
  logLevel: "Normal"
dns:
  baseDomain: microshift.example.com
etcd:
  memoryLimitMB: 0
ingress:
  listenAddress:
    - ""
```

```
  ports:
    http: 80
    https: 443
  routeAdmissionPolicy:
    namespaceOwnership: InterNamespaceAllowed
  status: Managed
kubelet:
manifests:
  kustomizePaths:
    - /usr/lib/microshift/manifests
    - /usr/lib/microshift/manifests.d/*
    - /etc/microshift/manifests
    - /etc/microshift/manifests.d/*
network:
  clusterNetwork:
    - 10.42.0.0/16
  serviceNetwork:
    - 10.43.0.0/16
  serviceNodePortRange: 30000-32767
node:
  hostnameOverride: ""
  nodeIP: "" (2)
  nodeIPv6: ""
storage:
  driver: "" (3)
  optionalCsiComponents: (4)
    - ""
```

1. Calculated based on the address of the service network.
2. The IP address of the default route.
3. Default null value deploys Logical Volume Managed Storage (LVMS).
4. Default null value deploys **snapshot-controller** and **snapshot-webhook**.

1.2. USING CUSTOM SETTINGS

To create custom configurations, make a copy of the **config.yaml.default** file that is provided in the **/etc/microshift/** directory, renaming it **config.yaml**. Keep this file in the **/etc/microshift/** directory, and then you can change supported settings that are expected to override the defaults before starting or restarting MicroShift.

IMPORTANT

Restart MicroShift after changing any configuration settings to have them take effect. The **config.yaml** file is read only when MicroShift starts.

1.2.1. Separate restarts

Applications and other optional services used with your MicroShift cluster might also need to be restarted separately to apply configuration changes throughout the cluster. For example, when making changes to certain networking settings, you must stop and restart service and application pods to apply

those changes. See each procedure for the task you are completing for more information.

TIP

If you add all of the configurations you need at the same time, you can minimize system restarts.

1.2.2. Parameters and values for the MicroShift config.yaml file

The following table explains MicroShift configuration YAML parameters and valid values for each:

Table 1.1. MicroShift config.yaml parameters

Field	Type	Description
advertiseAddress	**string**	A string that specifies the IP address from which the API server is advertised to members of the cluster. The default value is calculated based on the address of the service network.
auditLog.maxFileAge	**number**	How long log files are kept before automatic deletion. The default value of **0** in the **maxFileAge** parameter means a log file is never deleted based on age. This value can be configured.
auditLog.maxFileSize	**number**	By default, when the **audit.log** file reaches the **maxFileSize** limit, the **audit.log** file is rotated and MicroShift begins writing to a new **audit.log** file. This value can be configured.
auditLog.maxFiles	**number**	The total number of log files kept. By default, MicroShift retains 10 log files. The oldest is deleted when an excess file is created. This value can be configured.
auditLog.profile	**Default**, **WriteRequestBodies**, **AllRequestBodies**, or **None**	Logs only metadata for read and write requests; does not log request bodies except for OAuth access token requests. If you do not specify this field, the **Default** profile is used.
namedCertificates	**list**	Defines externally generated certificates and domain names by using custom certificate authorities.

Field	Type	Description
namedCertificates.certPath	**path**	The full path to the certificate.
namedCertificates.keyPath	**path**	The full path to the certificate key.
namedCertificates.names	**list**	Optional. Add a list of explicit DNS names. Leading wildcards are allowed. If no names are provided, the implicit names are extracted from the certificates.
subjectAltNames	Fully qualified domain names (FQDNs), wildcards such as ***.domain.com**, or IP addresses	Subject Alternative Names for API server certificates. SANs indicate all of the domain names and IP addresses that are secured by a certificate.
debugging.logLevel	**Normal**, **Debug**, **Trace**, or **TraceAll**	Log verbosity. Default is **Normal**.
dns.baseDomain	**valid domain**	Base domain of the cluster. All managed DNS records are subdomains of this base.
etcd.memoryLimitMB	**number**	By default, **etcd** uses as much memory as needed to handle the load on the system. However, in memory constrained systems, it might be preferred or necessary to limit the amount of memory **etcd** can to use at a given time.
ingress.listenAddress	IP address, NIC name, or multiple	Value defaults to the entire network of the host. The valid configurable value is a list that can be either a single IP address or NIC name or multiple IP addresses and NIC names.
ingress.ports.http	**80**	Default port shown. Configurable. Valid value is a single, unique port in the 1-65535 range. The values of the **ports.http** and **ports.https** fields cannot be the same.

Field	Type	Description
ingress.ports.https	**443**	Default port shown. Configurable. Valid value is a single, unique port in the 1-65535 range. The values of the **ports.http** and **ports.https** fields cannot be the same.
ingress.routeAdmissionPolicy. namespaceOwnership	**Strict** or **InterNamespaceAllowed**	Describes how hostname claims across namespaces are handled. By default, allows routes to claim different paths of the same hostname across namespaces. Specifying **Strict** prevents routes in different namespaces from claiming the same hostname. If the value is deleted in a customized MicroShift **config.yaml**, the **InterNamespaceAllowed** value is automatically set.
ingress.status	**Managed** or **Removed**	Router status. Default is **Managed**.
kubelet	See the MicroShift low-latency instructions	Parameter for passthrough configuration of the kubelet node agent. Used for low-latency configuration. Default value is null.
manifests	**list of paths**	The locations on the file system to scan for **kustomization** files to use to load manifests. Set to a list of paths to scan only those paths. Set to an empty list to disable loading manifests. The entries in the list can be glob patterns to match multiple subdirectories. Default values are **/usr/lib/microshift/manifests**, **/usr/lib/microshift/manifests.d/**, **/etc/microshift/manifests**, and **/etc/microshift/manifests.d/**.

Field	Type	Description
network.clusterNetwork	IP address block	A block of IP addresses from which pod IP addresses are allocated. IPv4 is the default. Dual-stack entries are supported. The first entry in this field is immutable after MicroShift starts. Default range is **10.42.0.0/16**.
network.serviceNetwork	IP address block	A block of virtual IP addresses for Kubernetes services. IP address pool for services. IPv4 is the default. Dual-stack entries are supported. The first entry in this field is immutable after MicroShift starts. Default range is **10.43.0.0/16**.
network.serviceNodePortRange	**range**	The port range allowed for Kubernetes services of type **NodePort**. If not specified, the default range of 30000-32767 is used. Services without a **NodePort** specified are automatically allocated one from this range. This parameter can be updated after MicroShift starts.
node.hostnameOverride	**string**	The name of the node. The default value is the hostname. If non-empty, this string is used to identify the node instead of the hostname. This value is immutable after MicroShift starts.
node.nodeIP	IPv4 address	The IPv4 address of the node. The default value is the IP address of the default route.
nodeIPv6	IPv6 address	The IPv6 address for the node for dual-stack configurations. Cannot be configured in single stack for either IPv4 or IPv6. Default is an empty value or null.
storage.driver	**none** or **lvms**	Default value is empty. An empty value or null field defaults to LVMS deployment.

Field	Type	Description
storage.optionalCsiComponents	**array**	Default value is null or an empty array. A null or empty array defaults to deploying **snapshot-controller** and **snapshot-webhook**. Expected values are **csi-snapshot-controller**, **csi-snapshot-webhook**, or **none**. An entry of **none** is mutually exclusive with all other values.

1.2.3. Configuring the advertise address network flag

The **apiserver.advertiseAddress** flag specifies the IP address on which to advertise the API server to members of the cluster. This address must be reachable by the cluster. You can set a custom IP address here, but you must also add the IP address to a host interface. Customizing this parameter preempts MicroShift from adding a default IP address to the **br-ex** network interface.

IMPORTANT

If you customize the **advertiseAddress** IP address, make sure it is reachable by the cluster when MicroShift starts by adding the IP address to a host interface.

If unset, the default value is set to the next immediate subnet after the service network. For example, when the service network is **10.43.0.0/16**, the **advertiseAddress** is set to **10.44.0.0/32**.

1.2.4. Extending the port range for NodePort services

The **serviceNodePortRange** setting extends the port range available to NodePort services. This option is useful when specific standard ports under the **30000-32767** range need to be exposed. For example, if your device needs to expose the **1883/tcp** MQ Telemetry Transport (MQTT) port on the network because client devices cannot use a different port.

IMPORTANT

NodePorts can overlap with system ports, causing a malfunction of the system or MicroShift.

Consider the following when configuring the NodePort service ranges:

- Do not create any NodePort service without an explicit **nodePort** selection. When an explicit **nodePort** is not specified, the port is assigned randomly by the **kube-apiserver** and cannot be predicted.
- Do not create any NodePort service for any system service port, MicroShift port, or other services you expose on your device **HostNetwork**.
- Table one specifies ports to avoid when extending the port range:

  Table 1.2. Ports to avoid.

Port	Description
22/tcp	SSH port
80/tcp	OpenShift Router HTTP endpoint
443/tcp	OpenShift Router HTTPS endpoint
1936/tcp	Metrics service for the openshift-router, not exposed today
2379/tcp	etcd port
2380/tcp	etcd port
6443	kubernetes API
8445/tcp	openshift-route-controller-manager
9537/tcp	cri-o metrics
10250/tcp	kubelet
10248/tcp	kubelet healthz port
10259/tcp	kube scheduler

1.3. ADDITIONAL RESOURCES

- Checking Greenboot status

CHAPTER 2. CONFIGURING IPV6 SINGLE OR DUAL-STACK NETWORKING

You can use the IPv6 networking protocol in either single-stack or dual-stack networking modes.

2.1. IPV6 NETWORKING WITH MICROSHIFT

The MicroShift service defaults to IPv4 address families cluster-wide. However, IPv6 single-stack and IPv4/IPv6 dual-stack networking is available on supported platforms.

- When you set the values for IPv6 in the MicroShift configuration file and restart the service, settings managed by the OVN-Kubernetes network plugin are updated automatically.
- After migrating to dual-stack networking, both new and existing pods have dual-stack networking enabled.
- If you require cluster-wide IPv6 access, such as for the control plane and other services, use the following configuration examples. The MicroShift Multus Container Network Interface (CNI) plugin can enable IPv6 for pods.
- For dual-stack networking, each MicroShift cluster network and service network supports up to two values in the cluster and service network configuration parameters.

IMPORTANT

Plan for IPv6 before starting MicroShift for the first time. Switching a cluster to and from different IP families is not supported unless you are migrating a cluster from default single-stack to dual-stack networking.

If you configure your networking for either IPv6 single stack or IPv4/IPv6 dual stack, you must restart application pods and services. Otherwise pods and services remain configured with the default IP family.

2.2. CONFIGURING IPV6 SINGLE-STACK NETWORKING

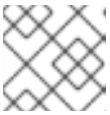
You can use the IPv6 network protocol by updating the MicroShift service configuration file.

Prerequisites

- You installed the OpenShift CLI (**oc**).
- You have root access to the cluster.
- Your cluster uses the OVN-Kubernetes network plugin.
- The host has an IPv6 address and IPv6 routes, including the default.

Procedure

1. If you have not done so, make a copy of the provided **config.yaml.default** file in the **/etc/microshift/** directory, renaming it **config.yaml**.
2. Keep the new MicroShift **config.yaml** in the **/etc/microshift/** directory. Your **config.yaml** file is read every time the MicroShift service starts.



NOTE

After you create it, the **config.yaml** file takes precedence over built-in settings.

3. Replace the default values in the **network** section of the MicroShift YAML with your valid values.

Example single-stack IPv6 networking configuration

```
apiServer:
# ...
network:
  clusterNetwork:
  - fd01::/48 1
  serviceNetwork:
  - fd02::/112 2
node:
  nodeIP: 2600:1f14:1c48:ee00:2d76:3190:5bc2:5aef 3
# ...
```

1. Specify a **clusterNetwork** with a CIDR value that is less than **64**.
2. Specify an IPv6 CIDR with a prefix of **112**. Kubernetes uses only the lowest 16 bits. For a prefix of **112**, IP addresses are assigned from **112** to **128** bits.
3. Example node IP address. Valid values are IP addresses in the IPv6 address family. You must only specify an IPv6 address when an IPv4 network is also present. If an IPv4 network is not present, the MicroShift service automatically fills in this value upon restart.

4. Complete any other configurations you require, then start MicroShift by running the following command:

```
$ sudo systemctl start microshift
```

Verification

1. Retrieve the networks defined in the node resource by running the following command:

```
$ oc get node -o jsonpath='{.items[].spec.podCIDRs[]}'
```

Example output

```
fd01::/48
```

2. Retrieve the status of the pods by running the following command:

```
$ oc get pod -A -o wide
```

Example output

```
NAMESPACE              NAME                                  READY   STATUS    RESTARTS
```

```
AGE   IP                  NODE          NOMINATED NODE   READINESS GATES
kube-system             csi-snapshot-controller-bb7cb654b-rqrt6   1/1     Running   0          65s
fd01:0:0:1::5          microshift-9   <none>          <none>
kube-system             csi-snapshot-webhook-95f475949-nbz8x      1/1     Running   0
61s   fd01:0:0:1::6          microshift-9   <none>          <none>
openshift-dns           dns-default-cjn66                         2/2     Running   0          62s
fd01:0:0:1::9          microshift-9   <none>          <none>
openshift-dns           node-resolver-ppnjb                       1/1     Running   0          63s
2001:db9:ca7:ff::1db8   microshift-9   <none>          <none>
openshift-ingress       router-default-6d97d7b8b6-wdtmg            1/1     Running   0          61s
fd01:0:0:1::8          microshift-9   <none>          <none>
openshift-ovn-kubernetes   ovnkube-master-gfvp5                   4/4     Running   0          63s
2001:db9:ca7:ff::1db8   microshift-9   <none>          <none>
openshift-ovn-kubernetes   ovnkube-node-bnpjh                     1/1     Running   0          63s
2001:db9:ca7:ff::1db8   microshift-9   <none>          <none>
openshift-service-ca    service-ca-5d7bd9db6-j25bd                1/1     Running   0          60s
fd01:0:0:1::4          microshift-9   <none>          <none>
openshift-storage       lvms-operator-656cd9b59b-bwr47            1/1     Running   0          63s
fd01:0:0:1::7          microshift-9   <none>          <none>
openshift-storage       vg-manager-f7dmk                          1/1     Running   0          27s
fd01:0:0:1::a          microshift-9   <none>          <none>
```

3. Retrieve the status of services by running the following command:

```
$ oc get svc -A
```

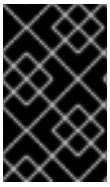
Example output

```
NAMESPACE          NAME                        TYPE          CLUSTER-IP   EXTERNAL-IP
PORT(S)                  AGE
default            kubernetes                  ClusterIP     fd02::1      <none>
443/TCP                  3m42s
kube-system        csi-snapshot-webhook        ClusterIP     fd02::4c4f   <none>
443/TCP                  3m20s
openshift-dns      dns-default                 ClusterIP     fd02::a      <none>
53/UDP,53/TCP,9154/TCP   2m58s
openshift-ingress  router-default              LoadBalancer  fd02::f2e6
2001:db9:ca7:ff::1db8,fd01:0:0:1::2,fd02::1:0,fd69::2   80:31133/TCP,443:31996/TCP
2m58s
openshift-ingress  router-internal-default     ClusterIP     fd02::c55e   <none>
80/TCP,443/TCP,1936/TCP  2m58s
openshift-storage  lvms-operator-metrics-service  ClusterIP     fd02::7afb   <none>
443/TCP                  2m58s
openshift-storage  lvms-webhook-service        ClusterIP     fd02::d8dd   <none>
443/TCP                  2m58s
openshift-storage  vg-manager-metrics-service   ClusterIP     fd02::fc1    <none>
443/TCP                  2m58s
```

2.3. CONFIGURING IPV6 DUAL-STACK NETWORKING BEFORE MICROSHIFT STARTS

You can configure your MicroShift cluster to run on dual-stack networking that supports IPv4 and IPv6 address families by using the configuration file before starting the service.

- The first IP family in the configuration is the primary IP stack in the cluster.
- After the cluster is running with dual-stack networking, enable application pods and add-on services for dual-stack by restarting them.



IMPORTANT

The OVN-Kubernetes network plugin requires that both IPv4 and IPv6 default routes be on the same network device. IPv4 and IPv6 default routes on separate network devices is not supported.

IMPORTANT

When using dual-stack networking where IPv6 is required, you cannot use IPv4-mapped IPv6 addresses, such as **::FFFF:198.51.100.1**.

Prerequisites

- You installed the OpenShift CLI (**oc**).
- You have root access to the cluster.
- Your cluster uses the OVN-Kubernetes network plugin.
- The host has both IPv4 and IPv6 addresses and routes, including a default for each.
- The host has at least two L3 networks, IPv4 and IPv6.

Procedure

1. If you have not done so, make a copy of the provided **config.yaml.default** file in the **/etc/microshift/** directory, renaming it **config.yaml**.
2. Keep the new MicroShift **config.yaml** in the **/etc/microshift/** directory. Your **config.yaml** file is read every time the MicroShift service starts.

NOTE

After you create it, the **config.yaml** file takes precedence over built-in settings.

3. If you have not started MicroShift, replace the default values in the **network** section of the MicroShift YAML with your valid values.

Example dual-stack IPv6 networking configuration with network assignments

```
apiServer:
# ...
apiServer:
  subjectAltNames:
  - 192.168.113.117
  - 2001:db9:ca7:ff::1db8
network:
  clusterNetwork:
  - 10.42.0.0/16
```

```
  - fd01::/48 1
  serviceNetwork:
  - 10.43.0.0/16
  - fd02::/112 2
node:
  nodeIP: 192.168.113.117 3
  nodeIPv6: 2001:db9:ca7:ff::1db8 4
# ...
```

1. Specify an IPv6 **clusterNetwork** with a CIDR value that is less than **64**.
2. Specify an IPv6 CIDR with a prefix of **112**. Kubernetes uses only the lowest 16 bits. For a prefix of **112**, IP addresses are assigned from **112** to **128** bits.
3. Example node IP address. Must be an IPv4 address family.
4. Example node IP address for dual-stack configuration. Must be an IPv6 address family. Configurable only with dual-stack networking.

4. Complete any other MicroShift configurations you require, then start MicroShift by running the following command:

```
$ sudo systemctl start microshift
```

5. Reset the IP family policy for application pods and services as needed, then restart those application pods and services to enable dual-stack networking. See "Resetting the IP family policy for application pods and services" for a simple example.

Verification

1. You can verify that all of the system services and pods to have two IP addresses, one for each family, by using the following steps:
   a. Retrieve the networks defined in the node resource by running the following command:

```
$ oc get pod -n openshift-ingress router-default-5b75594b4-w7w6s -o
jsonpath='{.status.podIPs}'
```

Example output

```
[{"ip":"10.42.0.4"},{"ip":"fd01:0:0:1::4"}]
```

   b. Retrieve the networks defined by the host network pods by running the following command:

```
$ oc get pod -n openshift-ovn-kubernetes ovnkube-master-2fm2k -o
jsonpath='{.status.podIPs}'
```

Example output

```
[{"ip":"192.168.113.117"},{"ip":"2001:db9:ca7:ff::1db8"}]
```

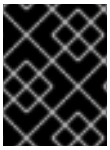
2.4. MIGRATING A MICROSHIFT CLUSTER TO IPV6 DUAL-STACK NETWORKING

You can convert a single-stack cluster to dual-stack cluster networking that supports IPv4 and IPv6 address families by setting two entries in the service and cluster network parameters in the MicroShift configuration file.

- The first IP family in the configuration is the primary IP stack in the cluster.
- MicroShift system pods and services are automatically updated upon MicroShift restart.
- After the cluster is migrated to dual-stack networking and has restarted, enable workload pods and services for dual-stack networking by restarting them.

IMPORTANT

The OVN-Kubernetes network plugin requires that both IPv4 and IPv6 default routes be on the same network device. IPv4 and IPv6 default routes on separate network devices is not supported.



IMPORTANT

When using dual-stack networking where IPv6 is required, you cannot use IPv4-mapped IPv6 addresses, such as **::FFFF:198.51.100.1**.

Prerequisites

- You installed the OpenShift CLI (**oc**).
- You have root access to the cluster.
- Your cluster uses the OVN-Kubernetes network plugin.
- The host has both IPv4 and IPv6 addresses and routes, including a default for each.
- The host has at least two L3 networks, IPv4 and IPv6.

Procedure

1. If you have not done so, make a copy of the provided **config.yaml.default** file in the **/etc/microshift/** directory, renaming it **config.yaml**.
2. Keep the new MicroShift **config.yaml** in the **/etc/microshift/** directory. Your **config.yaml** file is read every time the MicroShift service starts.

NOTE

After you create it, the **config.yaml** file takes precedence over built-in settings.

3. Add IPv6 configurations to the **network** section of the MicroShift YAML with your valid values:

WARNING

You must keep the same first entry across restarts and migrations. This is true for any migration: single-to-dual stack, or dual-to-single stack. A complete wipe of the etcd database is required if a change to the first entry is needed. This might result in application data loss and is not supported.

a. Add an IPv6 configuration for a second network in the **network** section of the MicroShift YAML with your valid values.

b. Add network assignments to the **network** section of the MicroShift **config.yaml** to enable dual stack with IPv6 as secondary network.

Example dual-stack IPv6 configuration with network assignments

```
# ...
apiServer:
  subjectAltNames:
  - 192.168.113.117
  - 2001:db9:ca7:ff::1db8 (1)
network:
  clusterNetwork:
  - 10.42.0.0/16 (2)
  - fd01::/48 (3)
  serviceNetwork:
  - 10.43.0.0/16
  - fd02::/112 (4)
node:
  nodeIP: 192.168.113.117 (5)
  nodeIPv6: 2001:db9:ca7:ff::1db8 (6)
# ...
```

(1) The IPv6 node address.

(2) IPv4 network. Specify a **clusterNetwork** with a CIDR value that is less than **24**.

(3) IPv6 network. Specify a **clusterNetwork** with a CIDR value that is less than **64**.

(4) Specify an IPv6 CIDR with a prefix of **112**. Kubernetes uses only the lowest 16 bits. For a prefix of **112**, IP addresses are assigned from **112** to **128** bits.

(5) Example node IP address. Maintain the previous IPv4 IP address.

(6) Example node IP address. Must be an IPv6 address family.

4. Complete any other configurations you require, then restart MicroShift by running the following command:

```
$ sudo systemctl restart microshift
```

5. Reset the IP family policy for application pods and services as needed, then restart those application pods and services to enable dual-stack networking. See "Resetting the IP family policy for application pods and services" for a simple example.

Verification

You can verify that all of the system services and pods to have two IP addresses, one for each family, by using the following steps:

1. Retrieve the status of the pods by running the following command:

```
$ oc get pod -A -o wide
```

Example output

```
NAMESPACE                NAME                                READY   STATUS    RESTARTS
AGE     IP              NODE          NOMINATED NODE   READINESS GATES
kube-system              csi-snapshot-controller-bb7cb654b-7s5ql   1/1     Running   0
46m     10.42.0.6       microshift-9   <none>          <none>
kube-system              csi-snapshot-webhook-95f475949-jrqv8      1/1     Running   0
46m     10.42.0.4       microshift-9   <none>          <none>
openshift-dns            dns-default-zxkqn                         2/2     Running   0             46m
10.42.0.5       microshift-9   <none>          <none>
openshift-dns            node-resolver-r2h5z                       1/1     Running   0             46m
192.168.113.117   microshift-9   <none>          <none>
openshift-ingress        router-default-5b75594b4-228z7            1/1     Running   0
2m5s    10.42.0.3       microshift-9   <none>          <none>
openshift-ovn-kubernetes   ovnkube-master-bltk7                    4/4     Running   2 (2m32s
ago)   2m36s   192.168.113.117   microshift-9   <none>          <none>
openshift-ovn-kubernetes   ovnkube-node-9ghgs                      1/1     Running   2 (2m32s
ago)   46m     192.168.113.117   microshift-9   <none>          <none>
openshift-service-ca       service-ca-5d7bd9db6-qgwgw              1/1     Running   0
46m     10.42.0.7       microshift-9   <none>          <none>
openshift-storage        lvms-operator-656cd9b59b-8rpf4            1/1     Running   0
46m     10.42.0.8       microshift-9   <none>          <none>
openshift-storage        vg-manager-wqmh4                          1/1     Running   2 (2m39s ago)
46m     10.42.0.10      microshift-9   <none>          <none>
```

2. Retrieve the networks defined by the OVN-K network plugin by running the following command:

```
$ oc get pod -n openshift-ovn-kubernetes ovnkube-master-bltk7 -o jsonpath='{.status.podIPs}'
```

Example output

```
[{"ip":"192.168.113.117"},{"ip":"2001:db9:ca7:ff::1db8"}]
```

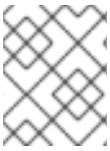
3. Retrieve the networks defined in the node resource by running the following command:

```
$ oc get pod -n openshift-ingress router-default-5b75594b4-228z7 -o
jsonpath='{.status.podIPs}'
```

Example output

```
[{"ip":"10.42.0.3"},{"ip":"fd01:0:0:1::3"}]
```

-



NOTE

To return to single-stack networking, you can remove the second entry to the networks and return to the single stack that was configured before migrating to dual-stack.

2.5. RESETTING THE IP FAMILY POLICY FOR APPLICATION PODS AND SERVICES

The default **ipFamilyPolicy** configuration value, **PreferSingleStack**, does not automatically update in all services after you update your MicroShift configuration to dual-stack networking. To enable dual-stack networking in services and application pods, you must update the **ipFamilyPolicy** value.

Prerequisites

- You used the MicroShift **config.yaml** to define a dual-stack network with an IPv6 address family.

Procedure

1. Set the **spec.ipFamilyPolicy** field to a valid value for dual-stack networking in your service or pod by using the following example:

   #### Example dual-stack network configuration for a service

   ```
   kind: Service
   apiVersion: v1
   metadata:
     name: microshift-new-service
     labels: app: microshift-application
   spec:
     type: NodePort
     ipFamilyPolicy: `PreferDualStack` 1
   # ...
   ```

   1 Required. Valid values for dual-stack networking are **PreferDualStack** and **RequireDualStack**. The value you set depends on the requirements of your application. **PreferSingleStack** is the default value for the **ipFamilyPolicy** field.

2. Restart any application pods that do not have a **hostNetwork** defined. Pods that do have a **hostNetwork** defined do not need to be restarted to update the **ipFamilyPolicy** value.

NOTE

MicroShift system services and pods are automatically updated when the **ipFamilyPolicy** value is updated.

2.6. OVN-KUBERNETES IPV6 AND DUAL-STACK LIMITATIONS

The OVN-Kubernetes network plugin has the following limitations:

- For a cluster configured for dual-stack networking, both IPv4 and IPv6 traffic must use the

same network interface as the default gateway. If this requirement is not met, pods on the host in the **ovnkube-node** daemon set enter the **CrashLoopBackOff** state. If you display a pod with a command such as **oc get pod -n openshift-ovn-kubernetes -l app=ovnkube-node -o yaml**, the **status** field contains more than one message about the default gateway, as shown in the following output:

```
I1006 16:09:50.985852   60651 helper_linux.go:73] Found default gateway interface br-ex
192.168.127.1
I1006 16:09:50.985923   60651 helper_linux.go:73] Found default gateway interface ens4
fe80::5054:ff:febe:bcd4
F1006 16:09:50.985939   60651 ovnkube.go:130] multiple gateway interfaces detected: br-ex
ens4
```

The only resolution is to reconfigure the host networking so that both IP families use the same network interface for the default gateway.

- For a cluster configured for dual-stack networking, both the IPv4 and IPv6 routing tables must contain the default gateway. If this requirement is not met, pods on the host in the **ovnkube-node** daemon set enter the **CrashLoopBackOff** state. If you display a pod with a command such as **oc get pod -n openshift-ovn-kubernetes -l app=ovnkube-node -o yaml**, the **status** field contains more than one message about the default gateway, as shown in the following output:

```
I0512 19:07:17.589083  108432 helper_linux.go:74] Found default gateway interface br-ex
192.168.123.1
F0512 19:07:17.589141  108432 ovnkube.go:133] failed to get default gateway interface
```

The only resolution is to reconfigure the host networking so that both IP families contain the default gateway.

2.7. ADDITIONAL RESOURCES

- Using NetworkManager to disable IPv6 for a specific connection (Red Hat Enterprise Linux documentation)

CHAPTER 3. CLUSTER ACCESS WITH KUBECONFIG

Learn about how **kubeconfig** files are used with MicroShift deployments. CLI tools use **kubeconfig** files to communicate with the API server of a cluster. These files provide cluster details, IP addresses, and other information needed for authentication.

3.1. KUBECONFIG FILES FOR CONFIGURING CLUSTER ACCESS

The two categories of **kubeconfig** files used in MicroShift are local access and remote access. Every time MicroShift starts, a set of **kubeconfig** files for local and remote access to the API server are generated. These files are generated in the **/var/lib/microshift/resources/kubeadmin/** directory using preexisting configuration information.

Each access type requires a different authentication certificate signed by different Certificate Authorities (CAs). The generation of multiple **kubeconfig** files accommodates this need.

You can use the appropriate **kubeconfig** file for the access type needed in each case to provide authentication details. The contents of MicroShift **kubeconfig** files are determined by either default built-in values or a **config.yaml** file.

NOTE

A **kubeconfig** file must exist for the cluster to be accessible. The values are applied from built-in default values or a **config.yaml**, if one was created.

Example contents of the kubeconfig files

```
/var/lib/microshift/resources/kubeadmin/
├── kubeconfig (1)
├── alt-name-1 (2)
│   └── kubeconfig
├── 1.2.3.4 (3)
│   └── kubeconfig
└── microshift-rhel9 (4)
    └── kubeconfig
```

1. Local host name. The main IP address of the host is always the default.
2. Subject Alternative Names for API server certificates.
3. DNS name.
4. MicroShift host name.

3.2. LOCAL ACCESS KUBECONFIG FILE

The local access **kubeconfig** file is written to **/var/lib/microshift/resources/kubeadmin/kubeconfig**. This **kubeconfig** file provides access to the API server using **localhost**. Choose this file when you are connecting the cluster locally.

Example contents of kubeconfig for local access

```
clusters:
- cluster:
    certificate-authority-data: <base64 CA>
    server: https://localhost:6443
```

The **localhost kubeconfig** file can only be used from a client connecting to the API server from the same host. The certificates in the file do not work for remote connections.

3.2.1. Accessing the MicroShift cluster locally

Use the following procedure to access the MicroShift cluster locally by using a **kubeconfig** file.

Prerequisites

- You have installed the **oc** binary.

Procedure

1. Optional: to create a **~/.kube/** folder if your Red Hat Enterprise Linux (RHEL) machine does not have one, run the following command:

```
$ mkdir -p ~/.kube/
```

2. Copy the generated local access **kubeconfig** file to the **~/.kube/** directory by running the following command:

```
$ sudo cat /var/lib/microshift/resources/kubeadmin/kubeconfig > ~/.kube/config
```

3. Update the permissions on your **~/.kube/config** file by running the following command:

```
$ chmod go-r ~/.kube/config
```

Verification

- Verify that MicroShift is running by entering the following command:

```
$ oc get all -A
```

3.3. REMOTE ACCESS KUBECONFIG FILES

When a MicroShift cluster connects to the API server from an external source, a certificate with all of the alternative names in the SAN field is used for validation. MicroShift generates a default **kubeconfig** for external access using the **hostname** value. The defaults are set in the **<node.hostnameOverride>**, **<node.nodeIP>** and **api.<dns.baseDomain>** parameter values of the default **kubeconfig** file.

The **/var/lib/microshift/resources/kubeadmin/<hostname>/kubeconfig** file uses the **hostname** of the machine, or **node.hostnameOverride** if that option is set, to reach the API server. The CA of the **kubeconfig** file is able to validate certificates when accessed externally.

Example contents of a default kubeconfig file for remote access

```
clusters:
```

```
- cluster:
    certificate-authority-data: <base64 CA>
    server: https://microshift-rhel9:6443
```

3.3.1. Remote access customization

Multiple remote access **kubeconfig** file values can be generated for accessing the cluster with different IP addresses or host names. An additional **kubeconfig** file generates for each entry in the **apiServer.subjectAltNames** parameter. You can copy remote access **kubeconfig** files from the host during times of IP connectivity and then use them to access the API server from other workstations.

3.4. GENERATING ADDITIONAL KUBECONFIG FILES FOR REMOTE ACCESS

You can generate additional **kubeconfig** files to use if you need more host names or IP addresses than the default remote access file provides.

IMPORTANT

You must restart MicroShift for configuration changes to be implemented.

Prerequisites

- You have created a **config.yaml** for MicroShift.

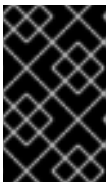
Procedure

1. Optional: You can show the contents of the **config.yaml**. Run the following command:

```
$ cat /etc/microshift/config.yaml
```

2. Optional: You can show the contents of the remote-access **kubeconfig** file. Run the following command:

```
$ cat /var/lib/microshift/resources/kubeadmin/<hostname>/kubeconfig
```



IMPORTANT

Additional remote access **kubeconfig** files must include one of the server names listed in the MicroShift **config.yaml** file. Additional **kubeconfig** files must also use the same CA for validation.

3. To generate additional **kubeconfig** files for additional DNS names SANs or external IP addresses, add the entries you need to the **apiServer.subjectAltNames** field. In the following example, the DNS name used is **alt-name-1** and the IP address is **1.2.3.4**.

Example config.yaml with additional authentication values

```
dns:
  baseDomain: example.com
node:
  hostnameOverride: "microshift-rhel9" (1)
```

```
  nodeIP: 10.0.0.1
apiServer:
  subjectAltNames:
  - alt-name-1 2
  - 1.2.3.4 3
```

1 Hostname

2 DNS name

3 IP address or range

4. Restart MicroShift to apply configuration changes and auto-generate the **kubeconfig** files you need by running the following command:

```
$ sudo systemctl restart microshift
```

5. To check the contents of additional remote-access **kubeconfig** files, insert the name or IP address as listed in the **config.yaml** into the **cat** command. For example, **alt-name-1** is used in the following example command:

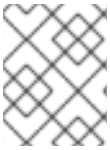
```
$ cat /var/lib/microshift/resources/kubeadmin/alt-name-1/kubeconfig
```

6. Choose the **kubeconfig** file to use that contains the SAN or IP address you want to use to connect your cluster. In this example, the **kubeconfig** containing `alt-name-1` in the **cluster.server** field is the correct file.

Example contents of an additional kubeconfig file

```
clusters:
- cluster:
    certificate-authority-data: <base64 CA>
    server: https://alt-name-1:6443 1
```

1 The **/var/lib/microshift/resources/kubeadmin/alt-name-1/kubeconfig** file values are from the **apiServer.subjectAltNames** configuration values.



NOTE

All of these parameters are included as common names (CN) and subject alternative names (SAN) in the external serving certificates for the API server.

3.4.1. Opening the firewall for remote access to the MicroShift cluster

Use the following procedure to open the firewall so that a remote user can access the MicroShift cluster. This procedure must be completed before a workstation user can access the cluster remotely.

For this procedure, **user@microshift** is the user on the MicroShift host machine and is responsible for setting up that machine so that it can be accessed by a remote user on a separate workstation.

Prerequisites

- You have installed the **oc** binary.
- Your account has cluster administration privileges.

Procedure

- As **user@microshift** on the MicroShift host, open the firewall port for the Kubernetes API server (**6443/tcp**) by running the following command:

```
[user@microshift]$ sudo firewall-cmd --permanent --zone=public --add-port=6443/tcp &&
sudo firewall-cmd --reload
```

Verification

- As **user@microshift**, verify that MicroShift is running by entering the following command:

```
[user@microshift]$ oc get all -A
```

3.4.2. Accessing the MicroShift cluster remotely

Use the following procedure to access the MicroShift cluster from a remote location by using a **kubeconfig** file.

The **user@workstation** login is used to access the host machine remotely. The **<user>** value in the procedure is the name of the user that **user@workstation** logs in with to the MicroShift host.

Prerequisites

- You have installed the **oc** binary.
- The **user@microshift** has opened the firewall from the local host.

Procedure

1. As **user@workstation**, create a **~/.kube/** folder if your Red Hat Enterprise Linux (RHEL) machine does not have one by running the following command:

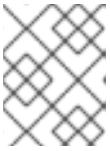
```
[user@workstation]$ mkdir -p ~/.kube/
```

2. As **user@workstation**, set a variable for the hostname of your MicroShift host by running the following command:

```
[user@workstation]$ MICROSHIFT_MACHINE=<name or IP address of MicroShift machine>
```

3. As **user@workstation**, copy the generated **kubeconfig** file that contains the host name or IP address you want to connect with from the RHEL machine running MicroShift to your local machine by running the following command:

```
[user@workstation]$ ssh <user>@$MICROSHIFT_MACHINE "sudo cat
/var/lib/microshift/resources/kubeadmin/$MICROSHIFT_MACHINE/kubeconfig" >
~/.kube/config
```



NOTE

To generate the **kubeconfig** files for this step, see Generating additional kubeconfig files for remote access.

4. As **user@workstation**, update the permissions on your **~/.kube/config** file by running the following command:

```
$ chmod go-r ~/.kube/config
```

Verification

- As **user@workstation**, verify that MicroShift is running by entering the following command:

```
[user@workstation]$ oc get all -A
```

CHAPTER 4. CONFIGURING CUSTOM CERTIFICATE AUTHORITIES

You can encrypt connections by using custom certificate authorities (CAs) with the MicroShift service.

4.1. HOW CUSTOM CERTIFICATE AUTHORITIES WORK IN MICROSHIFT

The default API server certificate is issued by an internal MicroShift cluster certificate authority (CA). Clients outside of the cluster cannot verify the API server certificate by default. This certificate can be replaced by a custom server certificate that is issued externally by a custom CA that clients trust. The following steps illustrate the workflow in MicroShift:

1. Copy the certificates and keys to the preferred directory in the host operating system. Ensure that the files are accessible by root only.

2. Update the MicroShift configuration for each custom CA by specifying the certificate names and new fully qualified domain name (FQDN) in the MicroShift **/etc/microshift/config.yaml** configuration file.
   Each certificate configuration can contain the following values:

   - The certificate file location is a required value.

   - A single common name containing the API server DNS and IP address or IP address range.

     TIP

     In most cases, MicroShift generates a new **kubeconfig** for your custom CA that includes the IP address or range that you specify. The exception is when wildcards are specified for the IP address. In this case, MicroShift generates a **kubeconfig** with the public IP address of the server. To use wildcards, you must update the **kubeconfig** file with your specific details.

   - Multiple Subject Alternative Names (SANs) containing the API server DNS and IP addresses or a wildcard certificate.

   - You can provide additional DNS names for each certificate.

3. After the MicroShift service restarts, you must copy the generated **kubeconfig** files to the client.

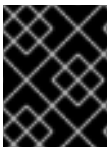
4. Configure additional CAs on the client system. For example, you can update CA bundles in the Red Hat Enterprise Linux (RHEL) truststore.

5. The certificates and keys are read from the specified file location on the host. Testing and validation of configuration is done from the client.

6. External server certificates are not automatically renewed. You must manually rotate your external certificates.

NOTE

If any validation fails, the MicroShift service skips the custom configuration and uses the default certificate to start. The priority is to continue the service uninterrupted. MicroShift logs errors when the service starts. Common errors include expired certificates, missing files, or incorrect IP addresses.



IMPORTANT

Custom server certificates have to be validated against CA data configured in the trust root of the host operating system. For information, see The system-wide truststore .

4.2. CONFIGURING CUSTOM CERTIFICATE AUTHORITIES

To configure externally generated certificates and domain names using custom certificate authorities (CAs), add them to the MicroShift **/etc/microshift/config.yaml** configuration file. You must also configure the host operating system trust root.

NOTE

Externally generated **kubeconfig** files are created in the **/var/lib/microshift/resources/kubeadmin/<hostname>/kubeconfig** directory. If you need to use **localhost** in addition to externally generated configurations, retain the original **kubeconfig** file in its default location. The **localhost kubeconfig** file uses the self-signed certificate authority.

Prerequisites

- The OpenShift CLI (**oc**) is installed.
- You have access to the cluster as a user with the cluster administration role.
- The certificate authority has issued the custom certificates.
- A MicroShift **/etc/microshift/config.yaml** configuration file exists.

Procedure

1. Copy the custom certificates you want to add to the trust root of the MicroShift host. Ensure that the certificate and private keys are only accessible to MicroShift.
2. For each custom CA that you need, add an **apiServer** section called **namedCertificates** to the **/etc/microshift/config.yaml** MicroShift configuration file by using the following example:

```
apiServer:
  namedCertificates:
  - certPath: ~/certs/api_fqdn_1.crt 1
    keyPath:  ~/certs/api_fqdn_1.key 2
  - certPath: ~/certs/api_fqdn_2.crt
    keyPath:  ~/certs/api_fqdn_2.key
    names: 3
    - api_fqdn_1
    - *.apps.external.com
```

1. Add the full path to the certificate.
2. Add the full path to the certificate key.
3. Optional. Add a list of explicit DNS names. Leading wildcards are allowed. If no names are provided, the implicit names are extracted from the certificates.

3. Restart the {microshift-service} to apply the certificates by running the following command:

```
$ systemctl microshift restart
```

4. Wait a few minutes for the system to restart and apply the custom server. New **kubeconfig** files are generated in the **/var/lib/microshift/resources/kubeadmin/** directory.

5. Copy the **kubeconfig** files to the client. If you specified wildcards for the IP address, update the **kubeconfig** to remove the public IP address of the server and replace that IP address with the specific wildcard range you want to use.

6. From the client, use the following steps:

   a. Specify the **kubeconfig** to use by running the following command:

```
$ export KUBECONFIG=~/custom-kubeconfigs/kubeconfig 1
```

   1 Use the location of the copied **kubeconfig** file as the path.

   b. Check that the certificates are applied by using the following command:

```
$ oc --certificate-authority ~/certs/ca.ca get node
```

   Example output

```
oc get node
NAME                          STATUS   ROLES                         AGE   VERSION
dhcp-1-235-195.arm.example.com   Ready    control-plane,master,worker   76m   v1.30.3
```

   c. Add the new CA file to the $KUBECONFIG environment variable by running the following command:

```
$ oc config set clusters.microshift.certificate-authority /tmp/certificate-authority-data-new.crt
```

   d. Verify that the new **kubeconfig** file contains the new CA by running the following command:

```
$ oc config view --flatten
```

   Example externally generated kubeconfig file

```
apiVersion: v1
clusters:
- cluster:
    certificate-authority: /tmp/certificate-authority-data-new.crt 1
    server: https://api.ci-ln-k0gim2b-76ef8.aws-2.ci.openshift.org:6443
  name: ci-ln-k0gim2b-76ef8
contexts:
- context:
    cluster: ci-ln-k0gim2b-76ef8
    user:
```

```
  name:
current-context:
kind: Config
preferences: {}
```

1 The **certificate-authority-data** section is not present in externally generated **kubeconfig** files. It is added with the **oc config set** command used previously.

e. Verify the **subject** and **issuer** of your customized API server certificate authority by running the following command:

```
$ curl --cacert /tmp/caCert.pem https://${fqdn_name}:6443/healthz -v
```

Example output

```
Server certificate:
  subject: CN=kas-test-cert_server
  start date: Mar 12 11:39:46 2024 GMT
  expire date: Mar 12 11:39:46 2025 GMT
  subjectAltName: host "dhcp-1-235-3.arm.eng.rdu2.redhat.com" matched cert's "dhcp-1-235-3.arm.eng.rdu2.redhat.com"
  issuer: CN=kas-test-cert_ca
  SSL certificate verify ok.
```

IMPORTANT

Either replace the **certificate-authority-data** in the generated **kubeconfig** file with the new **rootCA** or add the **certificate-authority-data** to the trust root of the operating system. Do not use both methods.

f. Configure additional CAs in the trust root of the operating system. For example, in the RHEL Client truststore on the client system. See The system-wide truststore for details.

- Updating the certificate bundle with the configuration that contains the CA is recommended.
- If you do not want to configure your certificate bundles, you can alternately use the **oc login localhost:8443 --certificate-authority=/path/to/cert.crt** command, but this method is not preferred.

4.3. CUSTOM CERTIFICATES RESERVED NAME VALUES

The following certificate problems cause MicroShift to ignore certificates dynamically and log an error:

- The certificate files do not exist on the disk or are not readable.
- The certificate is not parsable.
- The certificate overrides the internal certificates IP addresses or DNS names in a **SubjectAlternativeNames** (SAN) field. Do not use a reserved name when configuring SANs.

Table 4.1. Reserved Names values

Address	Type	Comment
localhost	DNS	
127.0.0.1	IP Address	
10.42.0.0	IP Address	Cluster Network
10.43.0.0/16,10.44.0.0/16	IP Address	Service Network
169.254.169.2/29	IP Address	br-ex Network
kubernetes.default.svc	DNS	
openshift.default.svc	DNS	
svc.cluster.local	DNS	

4.4. TROUBLESHOOTING CUSTOM CERTIFICATES

To troubleshoot the implementation of custom certificates, you can take the following steps.

Procedure

1. From MicroShift, ensure that the certificate is served by the **kube-apiserver** and verify that the certificate path is appended to the **--tls-sni-cert-key** FLAG by running the following command:

```
$ journalctl -u microshift -b0 | grep tls-sni-cert-key
```

Example output

```
Jan 24 14:53:00 localhost.localdomain microshift[45313]: kube-apiserver I0124
14:53:00.649099   45313 flags.go:64] FLAG: --tls-sni-cert-key="
[/home/eslutsky/dev/certs/server.crt,/home/eslutsky/dev/certs/server.key;/var/lib/microshift/certs/
kube-apiserver-external-signer/kube-external-serving/server.crt,/var/lib/microshift/certs/kube-
apiserver-external-signer/kube-external-serving/server.key;/var/lib/microshift/certs/kube-
apiserver-localhost-signer/kube-apiserver-localhost-
serving/server.crt,/var/lib/microshift/certs/kube-apiserver-localhost-signer/kube-apiserver-
localhost-serving/server.key;/var/lib/microshift/certs/kube-apiserver-service-network-
signer/kube-apiserver-service-network-serving/server.crt,/var/lib/microshift/certs/kube-
apiserver-service-network-signer/kube-apiserver-service-network-serving/server.key
```

2. From the client, ensure that the **kube-apiserver** is serving the correct certificate by running the following command:

```
$ openssl s_client -connect <SNI_ADDRESS>:6443 -showcerts | openssl x509 -text -noout -
in - | grep -C 1 "Alternative\|CN"
```

4.5. CLEANING UP AND RECREATING THE CUSTOM CERTIFICATES

To stop the MicroShift services, clean up the custom certificates and recreate the custom certificates, use the following steps.

Procedure

1. Stop the MicroShift services and clean up the custom certificates by running the following command:

```
$ sudo microshift-cleanup-data --cert
```

Example output

```
Stopping MicroShift services
Removing MicroShift certificates
MicroShift service was stopped
Cleanup succeeded
```

2. Restart the MicroShift services to recreate the custom certificates by running the following command:

```
$ sudo systemctl start microshift
```

4.6. ADDITIONAL RESOURCES

- OpenShift: Add an API server named certificate
- RHEL: Creating and managing TLS keys and certificates
- The system-wide truststore
- OpenShift CLI Reference: oc login

CHAPTER 5. CHECKING GREENBOOT SCRIPTS STATUS

To deploy applications or make other changes through the MicroShift API with tools other than **kustomize** manifests, you must wait until the greenboot health checks have finished. This ensures that your changes are not lost if greenboot rolls your **rpm-ostree** system back to an earlier state.

The **greenboot-healthcheck** service runs one time and then exits. After greenboot has exited and the system is in a healthy state, you can proceed with configuration changes and deployments.

5.1. CHECKING THE STATUS OF GREENBOOT HEALTH CHECKS

Check the status of greenboot health checks before making changes to the system or during troubleshooting. You can use any of the following commands to help you ensure that greenboot scripts have finished running.

Procedure

- To see a report of health check status, use the following command:

```
$ systemctl show --property=SubState --value greenboot-healthcheck.service
```

  - An output of **start** means that greenboot checks are still running.
  - An output of **exited** means that checks have passed and greenboot has exited. Greenboot runs the scripts in the **green.d** directory when the system is a healthy state.
  - An output of **failed** means that checks have not passed. Greenboot runs the scripts in **red.d** directory when the system is in this state and might restart the system.

- To see a report showing the numerical exit code of the service where **0** means success and non-zero values mean a failure occurred, use the following command:

```
$ systemctl show --property=ExecMainStatus --value greenboot-healthcheck.service
```

- To see a report showing a message about boot status, such as **Boot Status is GREEN - Health Check SUCCESS**, use the following command:

```
$ cat /run/motd.d/boot-status
```

CHAPTER 6. CONFIGURING AUDIT LOGGING POLICIES

You can control MicroShift audit log file rotation and retention by using configuration values.

6.1. ABOUT SETTING LIMITS ON AUDIT LOG FILES

Controlling the rotation and retention of the MicroShift audit log file by using configuration values helps keep the limited storage capacities of far-edge devices from being exceeded. On such devices, logging data accumulation can limit host system or cluster workloads, potentially causing the device stop working. Setting audit log policies can help ensure that critical processing space is continually available.

The values you set to limit MicroShift audit logs enable you to enforce the size, number, and age limits of audit log backups. Field values are processed independently of one another and without prioritization.

You can set fields in combination to define a maximum storage limit for retained logs. For example:

- Set both **maxFileSize** and **maxFiles** to create a log storage upper limit.
- Set a **maxFileAge** value to automatically delete files older than the timestamp in the file name, regardless of the **maxFiles** value.

6.1.1. Default audit log values

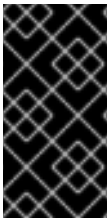
MicroShift includes the following default audit log rotation values:

Table 6.1. MicroShift default audit log values

Audit log parameter	Default setting	Definition
maxFileAge:	**0**	How long log files are retained before automatic deletion. The default value means that a log file is never deleted based on age. This value can be configured.
maxFiles:	**10**	The total number of log files retained. By default, MicroShift retains 10 log files. The oldest is deleted when an excess file is created. This value can be configured.
maxFileSize:	**200**	By default, when the **audit.log** file reaches the **maxFileSize** limit, the **audit.log** file is rotated and MicroShift begins writing to a new **audit.log** file. This value is in megabytes and can be configured.
profile:	**Default**	The **Default** profile setting only logs metadata for read and write requests; request bodies are not logged except for OAuth access token requests. If you do not specify this field, the **Default** profile is used.

The maximum default storage usage for audit log retention is 2000Mb if there are 10 or fewer files.

If you do not specify a value for a field, the default value is used. If you remove a previously set field value, the default value is restored after the next MicroShift service restart.



IMPORTANT

You must configure audit log retention and rotation in Red Hat Enterprise Linux (RHEL) for logs that are generated by application pods. These logs print to the console and are saved. Ensure that your log preferences are configured for the RHEL **/var/log/audit/audit.log** file to maintain MicroShift cluster health.

Additional resources

- Configuring auditd for a secure environment
- Understanding Audit log files
- How to use logrotate utility to rotate log files (Solutions, dated 7 August 2024)

6.2. ABOUT AUDIT LOG POLICY PROFILES

Audit log profiles define how to log requests that come to the OpenShift API server and the Kubernetes API server.

MicroShift supports the following predefined audit policy profiles:

Profile	Description
Default	Logs only metadata for read and write requests; does not log request bodies except for OAuth access token requests. This is the default policy.
WriteRequestBodies	In addition to logging metadata for all requests, logs request bodies for every write request to the API servers (**create**, **update**, **patch**, **delete**, **deletecollection**). This profile has more resource overhead than the **Default** profile. [1]
AllRequestBodies	In addition to logging metadata for all requests, logs request bodies for every read and write request to the API servers (**get**, **list**, **create**, **update**, **patch**). This profile has the most resource overhead.[1]
None	No requests are logged, including OAuth access token requests and OAuth authorize token requests. **WARNING** Do not disable audit logging by using the **None** profile unless you are fully aware of the risks of not logging data that can be beneficial when troubleshooting issues. If you disable audit logging and a support situation arises, you might need to enable audit logging and reproduce the issue to troubleshoot properly.

1. Sensitive resources, such as **Secret**, **Route**, and **OAuthClient** objects, are only logged at the metadata level.

By default, MicroShift uses the **Default** audit log profile. You can use another audit policy profile that also logs request bodies, but be aware of the increased resource usage such as CPU, memory, and I/O.

6.3. CONFIGURING AUDIT LOG VALUES

You can configure audit log settings by using the MicroShift service configuration file.

Procedure

1. Make a copy of the provided **config.yaml.default** file in the **/etc/microshift/** directory, renaming it **config.yaml**. Keep the new MicroShift **config.yaml** you create in the **/etc/microshift/** directory. The new **config.yaml** is read whenever the MicroShift service starts. After you create it, the **config.yaml** file takes precedence over built-in settings.

2. Replace the default values in the **auditLog** section of the YAML with your desired valid values.

   Example default auditLog configuration

```
apiServer:
# ....
  auditLog:
    maxFileAge: 7 (1)
    maxFileSize: 200 (2)
    maxFiles: 1 (3)
    profile: Default (4)
# ....
```

(1) Specifies the maximum time in days that log files are kept. Files older than this limit are deleted. In this example, after a log file is more than 7 days old, it is deleted. The files are deleted regardless of whether or not the live log has reached the maximum file size specified in the **maxFileSize** field. File age is determined by the timestamp written in the name of the rotated log file, for example, **audit-2024-05-16T17-03-59.994.log**. When the value is **0**, the limit is disabled.

(2) The maximum audit log file size in megabytes. In this example, the file is rotated as soon as the live log reaches the 200 MB limit. When the value is set to **0**, the limit is disabled.

(3) The maximum number of rotated audit log files retained. After the limit is reached, the log files are deleted in order from oldest to newest. In this example, the value **1** results in only 1 file of size **maxFileSize** being retained in addition to the current active log. When the value is set to **0**, the limit is disabled.

(4) Logs only metadata for read and write requests; does not log request bodies except for OAuth access token requests. If you do not specify this field, the **Default** profile is used.

3. Optional: To specify a new directory for logs, you can stop MicroShift, and then move the **/var/log/kube-apiserver** directory to your desired location:

   a. Stop MicroShift by running the following command:

```
$ sudo systemctl stop microshift
```

b. Move the **/var/log/kube-apiserver** directory to your desired location by running the following command:

```
$ sudo mv /var/log/kube-apiserver <~/kube-apiserver> 1
```

1 Replace ***<~/kube-apiserver>*** with the path to the directory that you want to use.

c. If you specified a new directory for logs, create a symlink to your custom directory at **/var/log/kube-apiserver** by running the following command:

```
$ sudo ln -s <~/kube-apiserver> /var/log/kube-apiserver 1
```

1 Replace ***<~/kube-apiserver>*** with the path to the directory that you want to use. This enables the collection of logs in sos reports.

4. If you are configuring audit log policies on a running instance, restart MicroShift by entering the following command:

```
$ sudo systemctl restart microshift
```

6.4. TROUBLESHOOTING AUDIT LOG CONFIGURATION

Use the following steps to troubleshoot custom audit log settings and file locations.

Procedure

- Check the current values that are configured by running the following command:

```
$ sudo microshift show-config --mode effective
```

Example output

```
auditLog:
  maxFileSize: 200
  maxFiles: 1
  maxFileAge: 7
  profile: AllRequestBodies
```

- Check the **audit.log** file permissions by running the following command:

```
$ sudo ls -ltrh /var/log/kube-apiserver/audit.log
```

Example output

```
-rw-------. 1 root root 46M Mar 12 09:52 /var/log/kube-apiserver/audit.log
```

- List the contents of the current log directory by running the following command:

```
$ sudo ls -ltrh /var/log/kube-apiserver/
```

Example output

```
total 6.0M
-rw-------. 1 root root 2.0M Mar 12 10:56 audit-2024-03-12T14-56-16.267.log
-rw-------. 1 root root 2.0M Mar 12 10:56 audit-2024-03-12T14-56-49.444.log
-rw-------. 1 root root 962K Mar 12 10:57 audit.log
```

CHAPTER 7. DISABLING THE LVMS CSI PROVIDER OR CSI SNAPSHOT

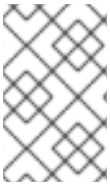
You can configure MicroShift to disable the built-in logical volume manager storage (LVMS) Container Storage Interface (CSI) provider or the CSI snapshot capabilities to reduce the use of runtime resources such as RAM, CPU, and storage.

7.1. DISABLING DEPLOYMENTS THAT RUN CSI SNAPSHOT IMPLEMENTATIONS

Use the following procedure to disable installation of the CSI implementation pods.

IMPORTANT

This procedure is for users who are defining the configuration file before installing and running MicroShift. If MicroShift is already started then CSI snapshot implementation will be running. Users must manually remove it by following the uninstallation instructions.



NOTE

MicroShift will not delete CSI snapshot implementation pods. You must configure MicroShift to disable installation of the CSI snapshot implementation pods during the startup process.

Procedure

1. Disable installation of the CSI snapshot controller by entering the **optionalCsiComponents** value under the **storage** section of the MicroShift configuration file in **/etc/microshift/config.yaml**:

```
# ...
  storage: {} 1
# ...
```

1 Accepted values are:

- Not defining **optionalCsiComponents**.
- Specifying **optionalCsiComponents** field with an empty value (**[]**) or a single empty string element (**[""]**).
- Specifying **optionalCsiComponents** with one of the accepted values which are **snapshot-controller**, **snapshot-webhook**, or **none**. **none** is mutually exclusive with all other values.

NOTE

If the **optionalCsiComponents** value is empty or null, MicroShift defaults to deploying snapshot-controller and snapshot-webhook.

2. After the **optionalCsiComponents** field is specified with a supported value in the **config.yaml**, start MicroShift by running the following command:

```
$ sudo systemctl start microshift
```

NOTE

MicroShift does not redeploy the disabled components after a restart.

7.2. DISABLING DEPLOYMENTS THAT RUN THE CSI DRIVER IMPLEMENTATIONS

Use the following procedure to disable installation of the CSI implementation pods.

IMPORTANT

This procedure is for users who are defining the configuration file before installing and running MicroShift. If MicroShift is already started then CSI driver implementation will be running. Users must manually remove it by following the uninstallation instructions.

NOTE

MicroShift will not delete CSI driver implementation pods. You must configure MicroShift to disable installation of the CSI driver implementation pods during the startup process.

Procedure

1. Disable installation of the CSI driver by entering the **driver** value under the **storage** section of the MicroShift configuration file in **/etc/microshift/config.yaml**:

```
# ...
  storage
   driver:
   - "none" 1
# ...
```

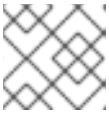
1 Valid values are **none** or **lvms**.

NOTE

By default, the **driver** value is empty or null and LVMS is deployed.

2. Start MicroShift after the **driver** field is specified with a supported value in the **/etc/microshift/config.yaml** file by running the following command:

```
$ sudo systemctl enable --now microshift
```



NOTE

MicroShift does not redeploy the disabled components after a restart operation.

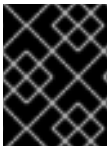
CHAPTER 8. CONFIGURING LOW LATENCY

8.1. CONFIGURING LOW LATENCY

You can configure and tune low latency capabilities to improve application performance on edge devices.

8.1.1. Lowering latency in MicroShift applications

Latency is defined as the time from an event to the response to that event. You can use low latency configurations and tuning in a MicroShift cluster running in an operational or software-defined control system where an edge device has to respond quickly to an external event. You can fully optimize low latency performance by combining MicroShift configurations with operating system tuning and workload partitioning.



IMPORTANT

The CPU set for management applications, such as the MicroShift service, OVS, CRI-O, MicroShift pods, and isolated cores, must contain all-online CPUs.

8.1.1.1. Workflow for configuring low latency for MicroShift applications

To configure low latency for applications running in a MicroShift cluster, you must complete the following tasks:

Required

- Install the **microshift-low-latency** RPM.
- Configure workload partitioning.
- Configure the **kubelet** section of the **config.yaml** file in the **/etc/microshift/** directory.
- Configure and activate a TuneD profile. TuneD is a Red Hat Enterprise Linux (RHEL) service that monitors the host system and optimizes performance under certain workloads.
- Restart the host.

Optional

- If you are using the x86_64 architecture, you can install Red Hat Enterprise Linux for Real Time 9.

Additional resources

- About low latency (OpenShift Container Platform documentation)

8.1.2. Installing the MicroShift low latency RPM package

When you install MicroShift, the low latency RPM package is not installed by default. You can install the low latency RPM as an optional package.

Prerequisites

1. You installed the MicroShift RPM.
2. You configured workload partitioning for MicroShift.

Procedure

- Install the low latency RPM package by running the following command:

```
$ sudo dnf install -y microshift-low-latency
```

TIP

Wait to restart the host until after activating your TuneD profile. Restarting the host restarts MicroShift and CRI-O, which applies the low latency manifests and activates the TuneD profile.

Next steps

1. Configure the kubelet parameter for low latency in the MicroShift **config.yaml**.
2. Tune your operating system, for example, configure and activate a TuneD profile.
3. Optional: Configure automatic activation of your TuneD profile.
4. Optional: If you are using the x86_64 architecture, install Red Hat Enterprise Linux for Real Time (real-time kernel).
5. Prepare your workloads for low latency.

8.1.3. Configuration kubelet parameters and values in MicroShift

The first step in enabling low latency to a MicroShift cluster is to add configurations to the MicroShift **config.yaml** file.

Prerequisites

- You installed the OpenShift CLI (**oc**).
- You have root access to the cluster.
- You made a copy of the provided **config.yaml.default** file in the **/etc/microshift/** directory, and renamed it **config.yaml**.

Procedure

- Add the kubelet configuration to the MicroShift **config.yaml** file:

Example passthrough kubelet configuration

```
apiServer:
# ...
kubelet: 1
  cpuManagerPolicy: static 2
  cpuManagerPolicyOptions:
    full-pcpus-only: "true" 3
```

```
  cpuManagerReconcilePeriod: 5s
  memoryManagerPolicy: Static 4
  topologyManagerPolicy: single-numa-node
  reservedSystemCPUs: 0-1 5
  reservedMemory:
  - limits:
      memory: 1100Mi 6
    numaNode: 0
  kubeReserved:
    memory: 500Mi
  systemReserved:
    memory: 500Mi
  evictionHard: 7
    imagefs.available: "15%" 8
    memory.available: "100Mi" 9
    nodefs.available: "10%" 10
    nodefs.inodesFree: "5%" 11
  evictionPressureTransitionPeriod: 0s
# ...
```

1. If you change the CPU or memory managers in the kubelet configuration, you must remove files that cache the previous configuration. Restart the host to remove them automatically, or manually remove the **/var/lib/kubelet/cpu_manager_state** and **/var/lib/kubelet/memory_manager_state** files.

2. The name of the policy to use. Valid values are **none** and **static**. Requires the **CPUManager** feature gate to be enabled. Default value is **none**.

3. A set of **key=value** pairs for setting extra options that fine tune the behavior of the **CPUManager** policies. The default value is **null**. Requires both the **CPUManager** and **CPUManagerPolicyOptions** feature gates to be enabled.

4. The name of the policy used by Memory Manager. Case-sensitive. The default value is **none**. Requires the **MemoryManager** feature gate to be enabled.

5. Required. The **reservedSystemCPUs** value must be the inverse of the offlined CPUs because both values combined must account for all of the CPUs on the system. This parameter is essential to dividing the management and application workloads. Use this parameter to define a static CPU set for the host-level system and Kubernetes daemons, plus interrupts and timers. Then the rest of the CPUs on the system can be used exclusively for workloads.

6. The value in **reservedMemory[0].limits.memory**, **1100** Mi in this example, is equal to **kubeReserved.memory** + **systemReserved.memory** + **evictionHard.memory.available**.

7. The **evictionHard** parameters define under which conditions the kubelet evicts pods. When you change the default value of only one parameter for the **evictionHard** stanza, the default values of other parameters are not inherited and are set to zero. Provide all the threshold values even when you want to change just one.

8. The **imagefs** is a filesystem that container runtimes use to store container images and container writable layers. In this example, the **evictionHard.imagefs.available** parameter means that the pod is evicted when the available space of the image filesystem is less than 15%.

9 In this example, the **evictionHard.memory.available** parameter means that the pods are evicted when the available memory of the node drops below 100MiB.

10 In this example, the **evictionHard.nodefs.available** parameter means that the pods are evicted when the main filesystem of the node has less than 10% available space.

11 In this example, the **evictionHard.nodefs.inodesFree** parameter means that the pods are evicted when more than 15% of the node's main filesystem's inodes are in use.

Verification

- After you complete the next steps and restart the host, you can use a root-access account to check that your settings are in the **config.yaml** file in the **/var/lib/microshift/resources/kubelet/config/** directory.

Next steps

1. Enable workload partitioning.
2. Tune your operating system. For example, configure and activate a TuneD profile.
3. Optional: Configure automatic enablement of your TuneD profile.
4. Optional: If you are using the x86_64 architecture, you can install Red Hat Enterprise Linux for Real Time (real-time kernel).
5. Prepare your MicroShift workloads for low latency.

Additional resources

- Using a YAML configuration file
- KubeletConfiguration reference (Kubernetes upstream documentation)

8.1.4. Tuning Red Hat Enterprise Linux 9

As a Red Hat Enterprise Linux (RHEL) system administrator, you can use the TuneD service to optimize the performance profile of RHEL for a variety of use cases. TuneD monitors and optimizes system performance under certain workloads, including latency performance.

- Use TuneD profiles to tune your system for different use cases, such as deploying a low-latency MicroShift cluster.
- You can modify the rules defined for each profile and customize tuning for a specific device.
- When you switch to another profile or deactivate TuneD, all changes made to the system settings by the previous profile revert back to their original state.
- You can also configure TuneD to react to changes in device usage and adjusts settings to improve performance of active devices and reduce power consumption of inactive devices.

8.1.4.1. Configuring the MicroShift TuneD profile

Configure a TuneD profile for your host to use low latency with MicroShift workloads using the **microshift-baseline-variables.conf** configuration file provided in the Red Hat Enterprise Linux (RHEL) **/etc/tuned/** host directory after you install the **microshift-low-latency** RPM package.

Prerequisites

- You have root access to the cluster.
- You installed the **microshift-low-latency** RPM package.
- Your RHEL host has TuneD installed. See Getting started with TuneD (RHEL documentation).

Procedure

1. You can use the default **microshift-baseline-variables.conf** TuneD profile in the **/etc/tuned/** directory profile, or create your own to add more tunings.

   Example microshift-baseline-variables.conf TuneD profile

```
# Isolate cores 2-7 for running application workloads
isolated_cores=2-7 (1)

# Size of the hugepages
hugepages_size=2M (2)

# Number of hugepages
hugepages=0

# Additional kernel arguments
additional_args= (3)

# CPU set to be offlined
offline_cpu_set= (4)
```

(1) Controls which cores should be isolated. By default, 1 core per socket is reserved in MicroShift for housekeeping. The other cores are isolated. Valid values are a core list or range. You can isolate any range, for example: **isolated_cores=2,4-7** or **isolated_cores=2-23**.

IMPORTANT

You must keep only one **isolated_cores=** variable.

NOTE

The Kubernetes CPU manager can use any CPU to run the workload except the reserved CPUs defined in the kubelet configuration. For this reason it is best that:

- The sum of the kubelet's reserved CPUs and isolated cores include all online CPUs.
- Isolated cores are complementary to the reserved CPUs defined in the kubelet configuration.

2. Size of the hugepages. Valid values are 2M or 1G.
3. Additional kernel arguments, for example, **additional_args=console=tty0 console=ttyS0,115200**.
4. The CPU set to be offlined.

IMPORTANT

Must not overlap with **isolated_cores**.

2. Enable the profile or make changes active, by running the following command:

```
$ sudo tuned-adm profile microshift-baseline
```

3. Reboot the host to make kernel arguments active.

Verification

- Optional: You can read the **/proc/cmdline** file that contains the arguments given to the currently running kernel on start.

```
$ cat /proc/cmdline
```

Example output

```
BOOT_IMAGE=(hd0,msdos2)/ostree/rhel-7f82ccd9595c3c70af16525470e32c6a81c9138c4eae6c79ab86d5a2d108d7fc/vmlinuz-5.14.0-427.31.1.el9_4.x86_64+rt crashkernel=1G-4G:192M,4G-64G:256M,64G-:512M rd.lvm.lv=rhel/root fips=0 console=ttyS0,115200n8 root=/dev/mapper/rhel-root rw ostree=/ostree/boot.1/rhel/7f82ccd9595c3c70af16525470e32c6a81c9138c4eae6c79ab86d5a2d108d7fc/0 skew_tick=1 tsc=reliable rcupdate.rcu_normal_after_boot=1 nohz=on nohz_full=2,4-5 rcu_nocbs=2,4-5 tuned.non_isolcpus=0000000b intel_pstate=disable nosoftlockup hugepagesz=2M hugepages=10
```

Next steps

1. Prepare your MicroShift workloads for low latency.
2. Optional: Configure automatic enablement of your TuneD profile.
3. Optional: If you are using the x86_64 architecture, you can install Red Hat Enterprise Linux for Real Time (real-time kernel).

Additional resources

- Getting started with TuneD (RHEL documentation)
- How to manage tuning profiles in Linux (Red Hat blog)

8.1.4.2. Automatically enable the MicroShift TuneD profile

Included in the **microshift-low-latency** RPM package is a systemd service that you can configure to automatically enable a TuneD profile when the system starts. This ability is particularly useful if you are installing MicroShift in a large fleet of devices.

Prerequisites

1. You installed the microshift-low-latency RPM package on the host.
2. You enabled low latency in the MicroShift **config.yaml**.
3. You created a TuneD profile.
4. You configured the **microshift-baseline-variables.conf** file.

Procedure

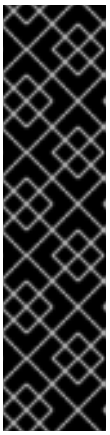
1. Configure the **tuned.yaml** in the **/etc/microshift/** directory, for example:

Example tuned.yaml

```
profile: microshift-baseline 1
reboot_after_apply: True 2
```

1 Controls which TuneD profile is activated. In this example, the name of the profile is **microshift-baseline**.

2 Controls whether the host must be rebooted after applying the profile. Valid values are **True** and **False**. For example, use the **True** setting to automatically restart the host after a new **ostree** commit is deployed.



IMPORTANT

The host is restarted when the **microshift-tuned.service** runs, but it does not restart the system when a new commit is deployed. You must restart the host to enable a new commit, then the system starts again when the **microshift-tuned.service** runs on that boot and detects changes to profiles and variables.

This double-boot can effect rollbacks. Ensure that you adjust the number of reboots in greenboot that are allowed before rollback when using automatic profile activation. For example, if 3 reboots are allowed before a rollback in greenboot, increase that number to 4. See the "Additional resources" list for more information.

2. Enable the **microshift-tuned.service** to run on each system start by entering the following command:

```
$ sudo systemctl enable microshift-tuned.service
```

IMPORTANT

If you set **reboot_after_apply** to **True**, ensure that a TuneD profile is active and that no other profiles have been activated outside of the MicroShift service. Otherwise, starting the **microshift-tuned.service** results in a host reboot.

3. Start the **microshift-tuned.service** by running the following command:

```
$ sudo systemctl start microshift-tuned.service
```

NOTE

The **microshift-tuned.service** uses collected checksums to detect changes to selected TuneD profiles and variables. If there are no checksums on the disk, the service activates the TuneD profile and restarts the host. Expect a host restart when first starting the **microshift-tuned.service**.

Next steps

- Optional: If you are using the x86_64 architecture, you can install Red Hat Enterprise Linux for Real Time (real-time kernel).

Additional resources

- Greenboot directories details

8.1.5. Using Red Hat Enterprise Linux for Real Time

If your workload has stringent low-latency determinism requirements for core kernel features such as interrupt handling and process scheduling in the microsecond (µs) range, you can use the Red Hat Enterprise Linux for Real Time (real-time kernel). The goal of the real-time kernel is consistent, low-latency determinism that offers predictable response times.

When considering system tuning, consider the following factors:

- System tuning is just as important when using the real-time kernel as it is for the standard kernel.
- Installing the real-time kernel on an untuned system running the standard kernel supplied as part of the RHEL 9 release is not likely to result in any noticeable benefit.
- Tuning the standard kernel yields 90% of possible latency gains.
- The real-time kernel provides the last 10% of latency reduction required by the most demanding workloads.

8.1.5.1. Installing the Red Hat Enterprise Linux for Real Time (real-time kernel)

Although the real-time kernel is not necessary for low latency workloads, using the real-time kernel can optimize low latency performance. You can install it on a host using RPM packages, and include it in a Red Hat Enterprise Linux for Edge (RHEL for Edge) image deployment.

Prerequisites

- You have a Red Hat subscription that includes Red Hat Enterprise Linux for Real Time (real-time kernel). For example, your host machine is registered and Red Hat Enterprise Linux (RHEL) is attached to a RHEL for Real Time subscription.
- You are using x86_64 architecture.

Procedure

1. Enable the real-time kernel repository by running the following command:

```
$ sudo subscription-manager repos --enable rhel-9-for-x86_64-rt-rpms
```

2. Install the real-time kernel by running the following command:

```
$ sudo dnf install -y kernel-rt
```

3. Query the real-time kernel version by running the following command:

```
$ RTVER=$(rpm -q --queryformat '%{version}-%{release}.%{arch}' kernel-rt | sort | tail -1)
```

4. Make a persistent change in GRUB that designates the real-time kernel as the default kernel by running the following command:

```
$ sudo grubby --set-default="/boot/vmlinuz-${RTVER}+rt"
```

5. Restart the host to activate the real-time kernel.

Next steps

1. Prepare your MicroShift workloads for low latency.
2. Optional: Use a blueprint to install the real-time kernel in a RHEL for Edge image.

8.1.5.2. Installing the Red Hat Enterprise Linux for Real Time (real-time kernel) in a Red Hat Enterprise Linux for Edge (RHEL for Edge) image

You can include the real-time kernel in a RHEL for Edge image deployment using image builder. The following example blueprint sections include references gathered from the previous steps required to configure low latency for a MicroShift cluster.

Prerequisites

- You have a Red Hat subscription enabled on the host that includes Red Hat Enterprise Linux for Real Time (real-time kernel).
- You are using the x86_64 architecture.
- You configured **osbuild** to use the **kernel-rt** repository.

IMPORTANT

A subscription that includes the real-time kernel must be enabled on the host used to build the commit.

Procedure

- Add the following example blueprint sections to your complete installation blueprint for installing the real-time kernel in a RHEL for Edge image:

Example blueprint snippet for the real-time kernel

```
[[packages]]
name = "microshift-low-latency"
version = "*"

# Kernel RT is supported only on the x86_64 architecture
[customizations.kernel]
name = "kernel-rt"

[customizations.services]
enabled = ["microshift", "microshift-tuned"]

[[customizations.files]]
path = "/etc/microshift/config.yaml"
data = """
kubelet:
  cpuManagerPolicy: static
  cpuManagerPolicyOptions:
    full-pcpus-only: "true"
  cpuManagerReconcilePeriod: 5s
  memoryManagerPolicy: Static
  topologyManagerPolicy: single-numa-node
  reservedSystemCPUs: 0-1
  reservedMemory:
  - limits:
      memory: 1100Mi
    numaNode: 0
  kubeReserved:
    memory: 500Mi
  systemReserved:
    memory: 500Mi
  evictionHard:
    imagefs.available: 15%
    memory.available: 100Mi
    nodefs.available: 10%
    nodefs.inodesFree: 5%
  evictionPressureTransitionPeriod: 0s
"""

[[customizations.files]]
path = "/etc/tuned/microshift-baseline-variables.conf"
data = """
# Isolated cores should be complementary to the kubelet configuration reserved CPUs.
# Isolated and reserved CPUs must contain all online CPUs.
# Core #3 is for testing offlining, therefore it is skipped.
isolated_cores=2,4-5
hugepages_size=2M
hugepages=10
additional_args=test1=on test2=true dummy
offline_cpu_set=3
"""

[[customizations.files]]
path = "/etc/microshift/tuned.yaml"
data = """
```

```
profile: microshift-baseline
reboot_after_apply: True
"""
```

Next steps

1. Complete the image building process.
2. If you have not completed the previous steps for enabling low latency for your MicroShift cluster, do so now. Update the blueprint with the information gathered in those steps.
3. If you have not configured workload partitioning, do so now.
4. Prepare your MicroShift workloads for low latency.

8.1.6. Building the Red Hat Enterprise Linux for Edge (RHEL for Edge) image with the real-time kernel

Complete the build process by starting with the following procedure to embed MicroShiftin a RHEL for Edge image. Then complete the remaining steps in the installation documentation for installing MicroShift in a RHEL for Edge image:

- Embedding in a RHEL for Edge image

Additional resources

- Red Hat Enterprise Linux for Real Time 9 (RHEL documentation)
- Using repositories that require subscription (osbuild documentation)
- Building RHEL images by using the real-time kernel for more information.
- Post installation instructions (RHEL for Real Time documentation)
- Embedding in a RHEL for Edge image
- FAQ about RHEL for Real Time (kernel-rt)

8.1.7. Preparing a MicroShift workload for low latency

To take advantage of low latency, workloads running on MicroShift must have the **microshift-low-latency** container runtime configuration set by using the **RuntimeClass** feature. The CRI-O **RuntimeClass** object is installed with the **microshift-low-latency** RPM, so only the pod annotations need to be configured.

Prerequisites

- You installed the **microshift-low-latency** RPM package.
- You configured workload partitioning.

Procedure

- Use the following example to set the following annotations in the pod spec:

```
cpu-load-balancing.crio.io: "disable"
irq-load-balancing.crio.io: "disable"
cpu-quota.crio.io: "disable"
cpu-load-balancing.crio.io: "disable"
cpu-freq-governor.crio.io: "<governor>"
```

Example pod that runs oslat test:

```
apiVersion: v1
kind: Pod
metadata:
  name: oslat
  annotations:
    cpu-load-balancing.crio.io: "disable" (1)
    irq-load-balancing.crio.io: "disable" (2)
    cpu-quota.crio.io: "disable" (3)
    cpu-c-states.crio.io: "disable" (4)
    cpu-freq-governor.crio.io: "<governor>" (5)
spec:
  runtimeClassName: microshift-low-latency (6)
  containers:
  - name: oslat
    image: quay.io/container-perf-tools/oslat
    imagePullPolicy: Always
    resources:
      requests:
        memory: "400Mi"
        cpu: "2"
      limits:
        memory: "400Mi"
        cpu: "2"
    env:
    - name: tool
      value: "oslat"
    - name: manual
      value: "n"
    - name: PRIO
      value: "1"
    - name: delay
      value: "0"
    - name: RUNTIME_SECONDS
      value: "60"
    - name: TRACE_THRESHOLD
      value: ""
    - name: EXTRA_ARGS
      value: ""
    securityContext:
      privileged: true
      capabilities:
        add:
          - SYS_NICE
          - IPC_LOCK
```

(1) Disables the CPU load balancing for the pod.

2 Opts the pod out of interrupt handling (IRQ).

3 Disables the CPU completely fair scheduler (CFS) quota at the pod run time.

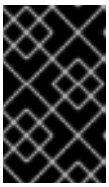
4 Enables or disables C-states for each CPU. Set the value to **disable** to provide the best performance for a high-priority pod.

5 Sets the **cpufreq** governor for each CPU. The **performance** governor is recommended for high-priority workloads.

6 The **runtimeClassName** must match the name of the performance profile configured in the cluster. For example, **microshift-low-latency**.

NOTE

Disable CPU load balancing only 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.



IMPORTANT

For the pod to have the **Guaranteed** QoS class, it must have the same values of CPU and memory in requests and limits. See Guaranteed (Kubernetes upstream documentation)

Additional resources

- Disabling power saving mode for high priority pods (Red Hat OpenShift Container Platform documentation)
- Disabling CPU CFS quota (Red Hat OpenShift Container Platform documentation)
- Disabling interrupt processing for CPUs where pinned containers are running (Red Hat OpenShift Container Platform documentation)

8.1.8. Reference blueprint for installing Red Hat Enterprise Linux for Real Time (real-time kernel) in a RHEL for Edge image

An image blueprint is a persistent definition of the required image customizations that enable you to create multiple builds. Instead of reconfiguring the blueprint for each image build, you can edit, rebuild, delete, and save the blueprint so that you can keep rebuilding images from it.

Example blueprint used to install the real-time kernel in a RHEL for Edge image

```
name = "microshift-low-latency"
description = "RHEL 9.4 and MicroShift configured for low latency"
version = "0.0.1"
modules = []
groups = []
distro = "rhel-94"

[[packages]]
name = "microshift"
```

```
version = "*"

[[packages]]
name = "microshift-greenboot"
version = "*"

[[packages]]
name = "microshift-networking"
version = "*"

[[packages]]
name = "microshift-selinux"
version = "*"

[[packages]]
name = "microshift-low-latency"
version = "*"

# Kernel RT is only available for x86_64
[customizations.kernel]
name = "kernel-rt"

[customizations.services]
enabled = ["microshift", "microshift-tuned"]

[customizations.firewall]
ports = ["22:tcp", "80:tcp", "443:tcp", "5353:udp", "6443:tcp", "30000-32767:tcp", "30000-32767:udp"]

[customizations.firewall.services]
enabled = ["mdns", "ssh", "http", "https"]

[[customizations.firewall.zones]]
name = "trusted"
sources = ["10.42.0.0/16", "169.254.169.1"]

[[customizations.files]]
path = "/etc/microshift/config.yaml"
data = """
kubelet:
  cpuManagerPolicy: static
  cpuManagerPolicyOptions:
    full-pcpus-only: "true"
  cpuManagerReconcilePeriod: 5s
  memoryManagerPolicy: Static
  topologyManagerPolicy: single-numa-node
  reservedSystemCPUs: 0-1
  reservedMemory:
  - limits:
      memory: 1100Mi
    numaNode: 0
  kubeReserved:
    memory: 500Mi
  systemReserved:
    memory: 500Mi
  evictionHard:
    imagefs.available: 15%
```

```
    memory.available: 100Mi
    nodefs.available: 10%
    nodefs.inodesFree: 5%
  evictionPressureTransitionPeriod: 0s
"""

[[customizations.files]]
path = "/etc/tuned/microshift-baseline-variables.conf"
data = """
# Isolated cores should be complementary to the kubelet configuration reserved CPUs.
# Isolated and reserved CPUs must contain all online CPUs.
# Core #3 is for testing offlining, therefore it is skipped.
isolated_cores=2,4-5
hugepages_size=2M
hugepages=10
additional_args=test1=on test2=true dummy
offline_cpu_set=3
"""

[[customizations.files]]
path = "/etc/microshift/tuned.yaml"
data = """
profile: microshift-baseline
reboot_after_apply: True
"""
```

Additional resources

- Workload partitioning

8.2. WORKLOAD PARTITIONING

Workload partitioning divides the node CPU resources into distinct CPU sets. The primary objective is to limit the amount of CPU usage for all control plane components which reserves rest of the device CPU resources for workloads of the user.

Workload partitioning allocates reserved set of CPUs to MicroShift services, cluster management workloads, and infrastructure pods, ensuring that the remaining CPUs in the cluster deployment are untouched and available exclusively for non-platform workloads.

8.2.1. Enabling workload partitioning

To enable workload partitioning on MicroShift, make the following configuration changes:

- Update the MicroShift **config.yaml** file to include the kubelet configuration file.
- Create the CRI-O systemd and configuration files.
- Create and update the systemd configuration file for the MicroShift and CRI-O services respectively.

Procedure

1. Update the MicroShift **config.yaml** file to include the kubelet configuration file to enable and configure CPU Manager for the workloads:

- Create the kubelet configuration file in the path **/etc/kubernetes/openshift-workload-pinning**. The kubelet configuration directs the kubelet to modify the node resources based on the capacity and allocatable CPUs.

  kubelet configuration example

```
# ...
{
  "management": {
    "cpuset": "0,6,7" (1)
  }
}
# ...
```

  (1) The **cpuset** applies to a machine with 8 VCPUs (4 cores) and is valid throughout the document.

- Update the MicroShift config.yaml file in the path **/etc/microshift/config.yaml**. Embed the kubelet configuration in the MicroShift **config.yaml** file to enable and configure CPU Manager for the workloads.

  MicroShift config.yaml example

```
# ...
kubelet:
  reservedSystemCPUs: 0,6,7 (1)
  cpuManagerPolicy: static
  cpuManagerPolicyOptions:
    full-pcpus-only: "true" (2)
  cpuManagerReconcilePeriod: 5s
# ...
```

  (1) Exclusive cpuset for the system daemons and the interrupts/timers.

  (2) kubelet configuration sets the **CPUManagerPolicyOptions** option to **full-pcpus-only** to ensure allocation of whole cores to the containers workload.

2. Create the CRI-O systemd and configuration files:

- Create the CRI-O configuration file in the path **/etc/crio/crio.conf.d/20-microshift-workload-partition.conf** which overrides the default configuration that already exists in the **11-microshift-ovn.conf** file.

  CRI-O configuration example

```
# ...
[crio.runtime]
infra_ctr_cpuset = "0,6,7"

[crio.runtime.workloads.management]
activation_annotation = "target.workload.openshift.io/management"
```

```
annotation_prefix = "resources.workload.openshift.io"
resources = { "cpushares" = 0, "cpuset" = "0,6,7" }
# ...
```

- Create the systemd file for CRI-O in the path **/etc/systemd/system/crio.service.d/microshift-cpuaffinity.conf**.

  CRI-O systemd configuration example

```
# ...
[Service]
CPUAffinity=0,6,7
# ...
```

3. Create and update the systemd configuration file with **CPUAffinity** value for the MicroShift and CRI-O services:

   - Create the MicroShift services systemd file in the path **/etc/systemd/system/microshift.service.d/microshift-cpuaffinity.conf**. MicroShift will be pinned using the systemd **CPUAffinity** value.

     MicroShift services systemd configuration example

```
# ...
[Service]
CPUAffinity=0,6,7
# ...
```

   - Update the **CPUAffinity** value in the MicroShift ovs-vswitchd systemd file in the path **/etc/systemd/system/ovs-vswitchd.service.d/microshift-cpuaffinity.conf**.

     MicroShift ovs-vswitchd systemd configuration example

```
# ...
[Service]
CPUAffinity=0,6,7
# ...
```

   - Update the **CPUAffinity** value in the MicroShift ovsdb-server systemd file in the path **/etc/systemd/system/ovsdb-server.service.d/microshift-cpuaffinity.conf**

     MicroShift ovsdb-server systemd configuration example

```
# ...
[Service]
CPUAffinity=0,6,7
# ...
```