Red Hat build of MicroShift 4.12

Networking

Configuring and managing cluster networking

Last Updated: 2023-10-03

Red Hat build of MicroShift 4.12 Networking

Configuring and managing cluster networking

Abstract

This document provides instructions for configuring and managing your MicroShift cluster network, including DNS, ingress, and the Pod network.

Table of Contents

CHAPTER 1. UNDERSTANDING NETWORKING SETTINGS

Learn how to apply networking customization and default settings to Red Hat build of MicroShift deployments. Each node is contained to a single machine and single Red Hat build of MicroShift, so each deployment requires individual configuration, pods, and settings.

Cluster Administrators have several options for exposing applications that run inside a cluster to external traffic and securing network connections:

- A service such as NodePort
- API resources, such as **Ingress** and **Route**

By default, Kubernetes allocates each pod an internal IP address for applications running within the pod. Pods and their containers can have traffic between them, but clients outside the cluster do not have direct network access to pods except when exposed with a service such as NodePort.

1.1. ABOUT THE OVN-KUBERNETES NETWORK PLUGIN

OVN-Kubernetes is the default networking solution for Red Hat build of MicroShift deployments. OVN-Kubernetes is a virtualized network for pods and services that is based on Open Virtual Network (OVN). The OVN-Kubernetes Container Network Interface (CNI) plugin is the network plugin for the cluster. A cluster that uses the OVN-Kubernetes network plugin also runs Open vSwitch (OVS) on the node. OVN configures OVS on the node to implement the declared network configuration.

1.1.1. Network topology

OVN-Kubernetes provides an overlay-based networking implementation. This overlay includes an OVS-based implementation of Service and NetworkPolicy. The overlay network uses the geneve tunnel, so the pod maximum transmission unit (MTU) is set to smaller than that of the physical interface on the host to remove the tunnel header.

OVS runs as a systemd service on the Red Hat build of MicroShift node. The OVS RPM package is installed as a dependency to the **microshift-networking** RPM package. OVS is started immediately when the **microshift-networking** RPM is installed.

1.1.1.1. IP forward

The host network **sysctl net.ipv4.ip_forward** kernel parameter is automatically enabled by the **ovnkube-master** container when started. This is required to forward incoming traffic to the CNI. For example, accessing the NodePort service from outside of a cluster fails if **ip_forward** is disabled.

1.1.2. Network performance optimizations

By default, three performance optimizations are applied to OVS services to minimize resource consumption:

- CPU affinity to **ovs-vswitchd.service** and **ovsdb-server.service**
- **no-mlockall** to **openvswitch.service**
- Limit handler and **revalidator** threads to **ovs-vswitchd.service**

1.1.3. Network features

Networking features available with Red Hat build of MicroShift 4.12 include:

- Kubernetes network policy
- Dynamic node IP
- Cluster network on specified host interface
- Secondary gateway interface
- Dual stack

Networking features not available with Red Hat build of MicroShift 4.12:

- Egress IP/firewall/qos: disabled
- Hybrid networking: not supported
- IPsec: not supported
- Hardware offload: not supported

1.1.4. Red Hat build of MicroShift networking components and services overview

This brief overview describes networking components and their operation in Red Hat build of MicroShift. The **microshift-networking** RPM is a package that automatically pulls in any networking-related dependencies and systemd services to initialize networking, for example, the **microshift-ovs-init** systemd service.

NetworkManager
: NetworkManager is required to set up the initial gateway bridge on the Red Hat build of MicroShift node. The NetworkManager and **NetworkManager-ovs** RPM packages are installed as dependencies to the **microshift-networking** RPM package, which contains the necessary configuration files. NetworkManager in Red Hat build of MicroShift uses the **keyfile** plugin and is restarted after installation of the **microshift-networking** RPM package.

microshift-ovs-init
: The **microshift-ovs-init.service** is installed by the **microshift-networking** RPM package as a dependent systemd service to microshift.service. It is responsible for setting up the OVS gateway bridge.

OVN containers
: Two OVN-Kubernetes daemon sets are rendered and applied by Red Hat build of MicroShift.

  - **ovnkube-master** Includes the **northd**, **nbdb**, **sbdb** and **ovnkube-master** containers.
  - **ovnkube-node** The ovnkube-node includes the OVN-Controller container.
    After Red Hat build of MicroShift boots, the OVN-Kubernetes daemon sets are deployed in the **openshift-ovn-kubernetes** namespace.

Packaging
: OVN-Kubernetes manifests and startup logic are built into Red Hat build of MicroShift. The systemd services and configurations included in **microshift-networking** RPM are:

  - **/etc/NetworkManager/conf.d/microshift-nm.conf** for NetworkManager.service

- **/etc/systemd/system/ovs-vswitchd.service.d/microshift-cpuaffinity.conf** for ovs-vswitchd.service
- **/etc/systemd/system/ovsdb-server.service.d/microshift-cpuaffinity.conf**
- **/usr/bin/configure-ovs-microshift.sh** for microshift-ovs-init.service
- **/usr/bin/configure-ovs.sh** for microshift-ovs-init.service
- **/etc/crio/crio.conf.d/microshift-ovn.conf** for CRI-O service

1.1.5. Bridge mappings

Bridge mappings allow provider network traffic to reach the physical network. Traffic leaves the provider network and arrives at the **br-int** bridge. A patch port between **br-int** and **br-ex** then allows the traffic to traverse to and from the provider network and the edge network. Kubernetes pods are connected to the **br-int** bridge through virtual ethernet pair: one end of the virtual ethernet pair is attached to the pod namespace, and the other end is attached to the **br-int** bridge.

1.1.5.1. Primary gateway interface

You can specify the desired host interface name in the **ovn.yaml** config file as **gatewayInterface**. The specified interface is added in OVS bridge br-ex which acts as gateway bridge for the CNI network.

1.1.5.2. Secondary gateway interface

You can set up one additional host interface for cluster ingress and egress in the **ovn.yaml** config file. The additional interface is added in a second OVS bridge **br-ex1**. Cluster pod traffic directed to the additional host subnet is routed automatically based on the destination IP through br-ex1.

Either two or three OVS bridges are created based on the CNI configuration:

Default deployment

- The **externalGatewayInterface** in not specified in the **ovn.yaml** config file.
- Two OVS bridges, **br-ex** and **br-int**, are created.

Customized deployment

- The **externalGatewayInterface** is user-specified in the **ovn.yaml** config file.
- Three OVS bridges are created: **br-ex**, **br-ex1** and **br-int**.

The br-ex bridge is created by **microshift-ovs-init.service** or manually. The br-ex bridge contains statically programmed openflow rules which distinguish traffic to and from the host network (underlay) and the OVN network (overlay).

The **br-int** bridge is created by the **ovnkube-master** container. The **br-int** bridge contains dynamically programmed openflow rules which handle cluster network traffic.

1.2. CREATING AN OVN-KUBERNETES CONFIGURATION FILE

Red Hat build of MicroShift uses built-in default OVN-Kubernetes values if an OVN-Kubernetes configuration file is not created. You can write an OVN-Kubernetes configuration file to **/etc/microshift/ovn.yaml**. An example file is provided for your configuration.

Procedure

1. To create your **ovn.yaml** file, run the following command:

```
$ sudo cp /etc/microshift/ovn.yaml.default /etc/microshift/ovn.yaml
```

2. To list the contents of the configuration file you created, run the following command:

```
$ cat /etc/microshift/ovn.yaml.default
```

Example 'yaml' configuration file with default values

```
ovsInit:
  disableOVSInit: false
  gatewayInterface: "" (1)
  externalGatewayInterface: "" (2)
mtu: 1400
```

(1) The default value is an empty string that means "not-specified." The CNI network plugin auto-detects to interface with the default route.

(2) The default value is an empty string that means "disabled."

3. To customize your configuration, use the following table that lists the valid values you can use:

Table 1.1. Supported optional OVN-Kubernetes configurations for Red Hat build of MicroShift

Field	Type	Default	Description	Example
ovsInit.disableOVSInit	bool	false	Skip configuring OVS bridge **br-ex** in **microshift-ovs-init.service**	true [1]
ovsInit.gatewayInterface	Alpha	eth0	Ingress that is the API gateway	eth0
ovsInit.externalGatewayInterface	Alpha	eth1	Ingress routing external traffic to your services and pods inside the node	eth1

Field	Type	Default	Description	Example
mtu	uint32	1400	MTU value used for the pods	1300

1. The OVS bridge is required. When **disableOVSInit** is true, OVS bridge **br-ex** must be configured manually.

> **IMPORTANT**
>
> If you change the **mtu** configuration value in the **ovn.yaml** file, you must restart the host that Red Hat build of MicroShift is running on to apply the updated setting.

Example custom ovn.yaml configuration file

```
ovsInit:
  disableOVSInit: true
  gatewayInterface: eth0
  externalGatewayInterface: eth1
mtu: 1300
```

> **IMPORTANT**
>
> When **disableOVSInit** is set to true in the **ovn.yaml** config file, the **br-ex** OVS bridge must be manually configured.

1.3. RESTARTING THE OVNKUBE-MASTER POD

The following procedure restarts the **ovnkube-master** pod.

Prerequisites

- The OpenShift CLI (**oc**) is installed.
- Access to the cluster as a user with the **cluster-admin** role.
- A cluster installed on infrastructure configured with the OVN-Kubernetes network plugin.
- The KUBECONFIG environment variable is set.

Procedure

Use the following steps to restart the **ovnkube-master** pod.

1. Access the remote cluster by running the following command:

```
$ export KUBECONFIG=$PWD/kubeconfig
```

2. Find the name of the **ovnkube-master** pod that you want to restart by running the following command:

```
$ pod=$(oc get pods -n openshift-ovn-kubernetes | awk -F " " '/ovnkube-master/{print $1}')
```

3. Delete the **ovnkube-master** pod by running the following command:

```
$ oc -n openshift-ovn-kubernetes delete pod $pod
```

4. Confirm that a new **ovnkube-master** pod is running by using the following command:

```
$ oc get pods -n openshift-ovn-kubernetes
```

   The listing of the running pods shows a new **ovnkube-master** pod name and age.

1.4. DEPLOYING RED HAT BUILD OF MICROSHIFT BEHIND AN HTTP(S) PROXY

Deploy a Red Hat build of MicroShift cluster behind an HTTP(S) proxy when you want to add basic anonymity and security measures to your pods.

You must configure the host operating system to use the proxy service with all components initiating HTTP(S) requests when deploying Red Hat build of MicroShift behind a proxy.

All the user-specific workloads or pods with egress traffic, such as accessing cloud services, must be configured to use the proxy. There is no built-in transparent proxying of egress traffic in Red Hat build of MicroShift.

1.5. USING A PROXY IN THE CRI-O CONTAINER RUNTIME

To use an HTTP(S) proxy in **CRI-O**, you need to set the **HTTP_PROXY** and **HTTPS_PROXY** environment variables. You can also set the **NO_PROXY** variable to exclude a list of hosts from being proxied.

Procedure

1. Add the following settings to the **/etc/systemd/system/crio.service.d/00-proxy.conf** file:

```
Environment=NO_PROXY="localhost,127.0.0.1"
Environment=HTTP_PROXY="http://$PROXY_USER:$PROXY_PASSWORD@$PROXY_SE
RVER:$PROXY_PORT/"
Environment=HTTPS_PROXY="http://$PROXY_USER:$PROXY_PASSWORD@$PROXY_S
ERVER:$PROXY_PORT/"
```

2. Reload the configuration settings:

```
$ sudo systemctl daemon-reload
```

3. Restart the CRI-O service to apply the settings:

```
$ sudo systemctl restart crio
```

1.6. GETTING A SNAPSHOT OF OVS INTERFACES FROM A RUNNING CLUSTER

A snapshot represents the state and data of OVS interfaces at a specific point in time.

Procedure

- To see a snapshot of OVS interfaces from a running Red Hat build of MicroShift cluster, use the following command:

```
$ sudo ovs-vsctl show
```

Example OVS interfaces in a running cluster

```
9d9f5ea2-9d9d-4e34-bbd2-dbac154fdc93
    Bridge br-ex
        Port enp1s0
            Interface enp1s0
                type: system
        Port br-ex
            Interface br-ex
                type: internal
        Port patch-br-ex_localhost.localdomain-to-br-int 1
            Interface patch-br-ex_localhost.localdomain-to-br-int
                type: patch
                options: {peer=patch-br-int-to-br-ex_localhost.localdomain} 2
    Bridge br-int
        fail_mode: secure
        datapath_type: system
        Port patch-br-int-to-br-ex_localhost.localdomain
            Interface patch-br-int-to-br-ex_localhost.localdomain
                type: patch
                options: {peer=patch-br-ex_localhost.localdomain-to-br-int}
        Port eebee1ce5568761
            Interface eebee1ce5568761 3
        Port b47b1995ada84f4
            Interface b47b1995ada84f4 4
        Port "3031f43d67c167f"
            Interface "3031f43d67c167f" 5
        Port br-int
            Interface br-int
                type: internal
        Port ovn-k8s-mp0 6
            Interface ovn-k8s-mp0
                type: internal
    ovs_version: "2.17.3"
```

1 2 The **patch-br-ex_localhost.localdomain-to-br-int** and **patch-br-int-to-br-ex_localhost.localdomain** are OVS patch ports that connect **br-ex** and **br-int**.

3 4 5 The pod interfaces **eebee1ce5568761**, **b47b1995ada84f4** and **3031f43d67c167f** are named with the first 15 bits of pod sandbox ID and are plugged in the **br-int** bridge.

6 The OVS internal port for hairpin traffic,**ovn-k8s-mp0** is created by the **ovnkube-master** container.

1.7. THE MULTICAST DNS PROTOCOL

The multicast DNS protocol (mDNS) allows name resolution and service discovery within a Local Area Network (LAN) using multicast exposed on the **5353**/**UDP** port.

Red Hat build of MicroShift includes an embedded mDNS server for deployment scenarios in which the authoritative DNS server cannot be reconfigured to point clients to services on Red Hat build of MicroShift. The embedded DNS server allows **.local** domains exposed by Red Hat build of MicroShift to be discovered by other elements on the LAN.

Additional resources

- Troubleshooting
- Troubleshooting the NodePort service
- NodePort unreachable workround

CHAPTER 2. USING A FIREWALL

Firewalls are not required in Red Hat build of MicroShift, but using a firewall can prevent undesired access to the Red Hat build of MicroShift API.

2.1. ABOUT NETWORK TRAFFIC THROUGH THE FIREWALL

When using a firewall, you must explicitly allow the following OVN-Kubernetes traffic when the **firewalld** service is running:

CNI pod to CNI pod

CNI pod to Host-Network pod Host-Network pod to Host-Network pod

CNI pod

The Kubernetes pod that uses the CNI network

Host-Network pod

The Kubernetes pod that uses host network Install and configure the **firewalld** service by using the following procedures.

IMPORTANT

Red Hat build of MicroShift pods must have access to the internal CoreDNS component and API servers.

2.2. INSTALLING THE FIREWALLD SERVICE

Use the following procedure to install and run the **firewalld** service for Red Hat build of MicroShift.

Procedure

1. To install the **firewalld** service, run the following command:

```
$ sudo dnf install -y firewalld
```

2. To initiate the firewall, run the following command:

```
$ sudo systemctl enable firewalld --now
```

2.3. REQUIRED FIREWALL SETTINGS

An IP address range for the cluster network must be enabled during firewall configuration. You can use the default values or customize the IP address range. If you choose to customize the cluster network IP address range from the default **10.42.0.0/16** setting, you must also use the same custom range in the firewall configuration.

Table 2.1. Firewall IP address settings

IP Range	Firewall rule required	Description
10.42.0.0/16	No	Host network pod access to other pods

IP Range	Firewall rule required	Description
169.254.169.1	Yes	Host network pod access to Red Hat build of MicroShift API server

The following are examples of commands for settings that are mandatory for firewall configuration:

Example commands

- Configure host network pod access to other pods:

```
$ sudo firewall-cmd --permanent --zone=trusted --add-source=10.42.0.0/16
```

- Configure host network pod access to services backed by Host endpoints, such as the Red Hat build of MicroShift API:

```
$ sudo firewall-cmd --permanent --zone=trusted --add-source=169.254.169.1
```

2.4. USING OPTIONAL PORT SETTINGS

The Red Hat build of MicroShift firewall service allows optional port settings.

Procedure

- To add customized ports to your firewall configuration, use the following command syntax:

```
$ sudo firewall-cmd --permanent --zone=public --add-port=<port number>/<port protocol>
```

Table 2.2. Optional ports

Port(s)	Protocol(s)	Description
80	TCP	HTTP port used to serve applications through the OpenShift Container Platform router.
443	TCP	HTTPS port used to serve applications through the OpenShift Container Platform router.
5353	UDP	mDNS service to respond for OpenShift Container Platform route mDNS hosts.
30000-32767	TCP	Port range reserved for NodePort services; can be used to expose applications on the LAN.

Port(s)	Protocol(s)	Description
30000-32767	UDP	Port range reserved for NodePort services; can be used to expose applications on the LAN.
6443	TCP	HTTPS API port for the Red Hat build of MicroShift API.

The following are examples of commands used when requiring external access through the firewall to services running on Red Hat build of MicroShift, such as port 6443 for the API server, for example, ports 80 and 443 for applications exposed through the router.

Example commands

- Configuring a port for the Red Hat build of MicroShift API server:

```
$ sudo firewall-cmd --permanent --zone=public --add-port=6443/tcp
```

- Configuring ports for applications exposed through the router:

```
$ sudo firewall-cmd --permanent --zone=public --add-port=80/tcp
```

```
$ sudo firewall-cmd --permanent --zone=public --add-port=443/tcp
```

2.5. ALLOWING NETWORK TRAFFIC THROUGH THE FIREWALL

You can allow network traffic through the firewall by first configuring the IP address range with either default or custom values, and then allow internal traffic from pods through the network gateway by inserting the DNS server.

Procedure

Set the default values or a custom IP address range. After setting the IP address range, allow internal traffic from the pods through the network gateway.

1. To set the IP address range:

   a. To configure the IP address range with default values, run the following command:

   ```
   $ sudo firewall-offline-cmd --permanent --zone=trusted --add-source=10.42.0.0/16
   ```

   b. Alternatively, you can configure the IP address range with custom values by running the following command:

   ```
   $ sudo firewall-offline-cmd --permanent --zone=trusted --add-source=<custom IP range>
   ```

2. To allow internal traffic from pods through the network gateway, run the following command:

```
$ sudo firewall-offline-cmd --permanent --zone=trusted --add-source=169.254.169.1
```

2.5.1. Applying firewall settings

To apply firewall settings, use the following one-step procedure:

Procedure

After you have finished configuring network access through the firewall, run the following command to restart the firewall and apply settings:

```
$ sudo firewall-cmd --reload
```

2.6. VERIFYING FIREWALL SETTINGS

After you have restarted the firewall, you can verify your settings by listing them.

Procedure

- To verify rules added in the default public zone, such as ports-related rules, run the following command:

  ```
  $ sudo firewall-cmd --list-all
  ```

- To verify rules added in the trusted zone, such as IP-range related rules, run the following command:

  ```
  $ sudo firewall-cmd --zone=trusted --list-all
  ```

2.7. KNOWN FIREWALL ISSUE

- To avoid breaking traffic flows with a firewall reload or restart, execute firewall commands before starting Red Hat build of MicroShift. The CNI driver in Red Hat build of MicroShift makes use of iptable rules for some traffic flows, such as those using the NodePort service. The iptable rules are generated and inserted by the CNI driver, but are deleted when the firewall reloads or restarts. The absence of the iptable rules breaks traffic flows. If firewall commands have to be executed after Red Hat build of MicroShift is running, manually restart **ovnkube-master** pod in the **openshift-ovn-kubernetes** namespace to reset the rules controlled by the CNI driver.

Additional resources

- Troubleshooting iptables deleted.