Red Hat build of MicroShift 4.15

Networking

Configuring and managing cluster networking

Last Updated: 2024-04-08

Red Hat build of MicroShift 4.15 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. ABOUT THE OVN-KUBERNETES NETWORK PLUGIN

The OVN-Kubernetes Container Network Interface (CNI) plugin is the default networking solution for MicroShift clusters. OVN-Kubernetes is a virtualized network for pods and services that is based on Open Virtual Network (OVN).

- Default network configuration and connections are applied automatically in MicroShift with the **microshift-networking** RPM during installation.
- A cluster that uses the OVN-Kubernetes network plugin also runs Open vSwitch (OVS) on the node.
- OVN-K configures OVS on the node to implement the declared network configuration.
- Host physical interfaces are not bound by default to the OVN-K gateway bridge, **br-ex**. You can use standard tools on the host for managing the default gateway, such as the Network Manager CLI (**nmcli**).
- Changing the CNI is not supported on MicroShift.

Using configuration files or custom scripts, you can configure the following networking settings:

- You can use subnet CIDR ranges to allocate IP addresses to pods.
- You can change the maximum transmission unit (MTU) value.
- You can configure firewall ingress and egress.
- You can define network policies in the MicroShift cluster, including ingress and egress rules.

1.1. MICROSHIFT NETWORKING CUSTOMIZATION MATRIX

The following table summarizes the status of networking features and capabilities that are either present as defaults, supported for configuration, or not available with the MicroShift service:

Table 1.1. MicroShift networking capabilities and customization status

Network feature	Availability	Customization supported
Advertise address	Yes	Yes [1]
Kubernetes network policy	Yes	Yes
Kubernetes network policy logs	Not available	N/A
Load balancing	Yes	Yes
Multicast DNS	Yes	Yes [2]
Network proxies	Yes [3]	CRI-O

Network feature	Availability	Customization supported
Network performance	Yes	MTU configuration
Egress IPs	Not available	N/A
Egress firewall	Not available	N/A
Egress router	Not available	N/A
Firewall	No [4]	Yes
Hardware offloading	Not available	N/A
Hybrid networking	Not available	N/A
IPsec encryption for intra-cluster communication	Not available	N/A
IPv6	Not available [5]	N/A

1. 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**.
2. You can use the multicast DNS protocol (mDNS) to allow name resolution and service discovery within a Local Area Network (LAN) using multicast exposed on the **5353/UDP** port.
3. There is no built-in transparent proxying of egress traffic in MicroShift. Egress must be manually configured.
4. Setting up the firewalld service is supported by RHEL for Edge.
5. IPv6 is not available in any configuration.

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

```
dns:
  baseDomain: microshift.example.com (1)
network:
  clusterNetwork:
    - 10.42.0.0/16 (2)
  serviceNetwork:
    - 10.43.0.0/16 (3)
  serviceNodePortRange: 30000-32767 (4)
node:
  hostnameOverride: "" (5)
  nodeIP: "" (6)
apiServer:
  advertiseAddress: 10.44.0.0/32 (7)
  subjectAltNames: [] (8)
debugging:
  logLevel: "Normal" (9)
```

1. Base domain of the cluster. All managed DNS records will be subdomains of this base.
2. A block of IP addresses from which Pod IP addresses are allocated.
3. A block of virtual IP addresses for Kubernetes services.
4. The port range allowed for Kubernetes services of type NodePort.
5. The name of the node. The default value is the hostname.
6. The IP address of the node. The default value is the IP address of the default route.
7. 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.
8. Subject Alternative Names for API server certificates.
9. Log verbosity. Valid values for this field are **Normal**, **Debug**, **Trace**, or **TraceAll**.

1.2. NETWORK FEATURES

Networking features available with MicroShift 4.15 include:

- Kubernetes network policy
- Dynamic node IP
- Custom gateway interface
- Second gateway interface
- Cluster network on specified host interface
- Blocking external access to NodePort service on specific host interfaces

Networking features not available with MicroShift 4.15:

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

Additional resources

- Using a YAML configuration file
- Understanding networking settings

1.3. 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.

1.4. 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**

1.5. MICROSHIFT NETWORKING COMPONENTS AND SERVICES

This brief overview describes networking components and their operation in 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 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 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 MicroShift.

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

Packaging
: OVN-Kubernetes manifests and startup logic are built into MicroShift. The systemd services and configurations included in the **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** for **ovs-server.service**
- **/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 the CRI-O service

1.6. 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.

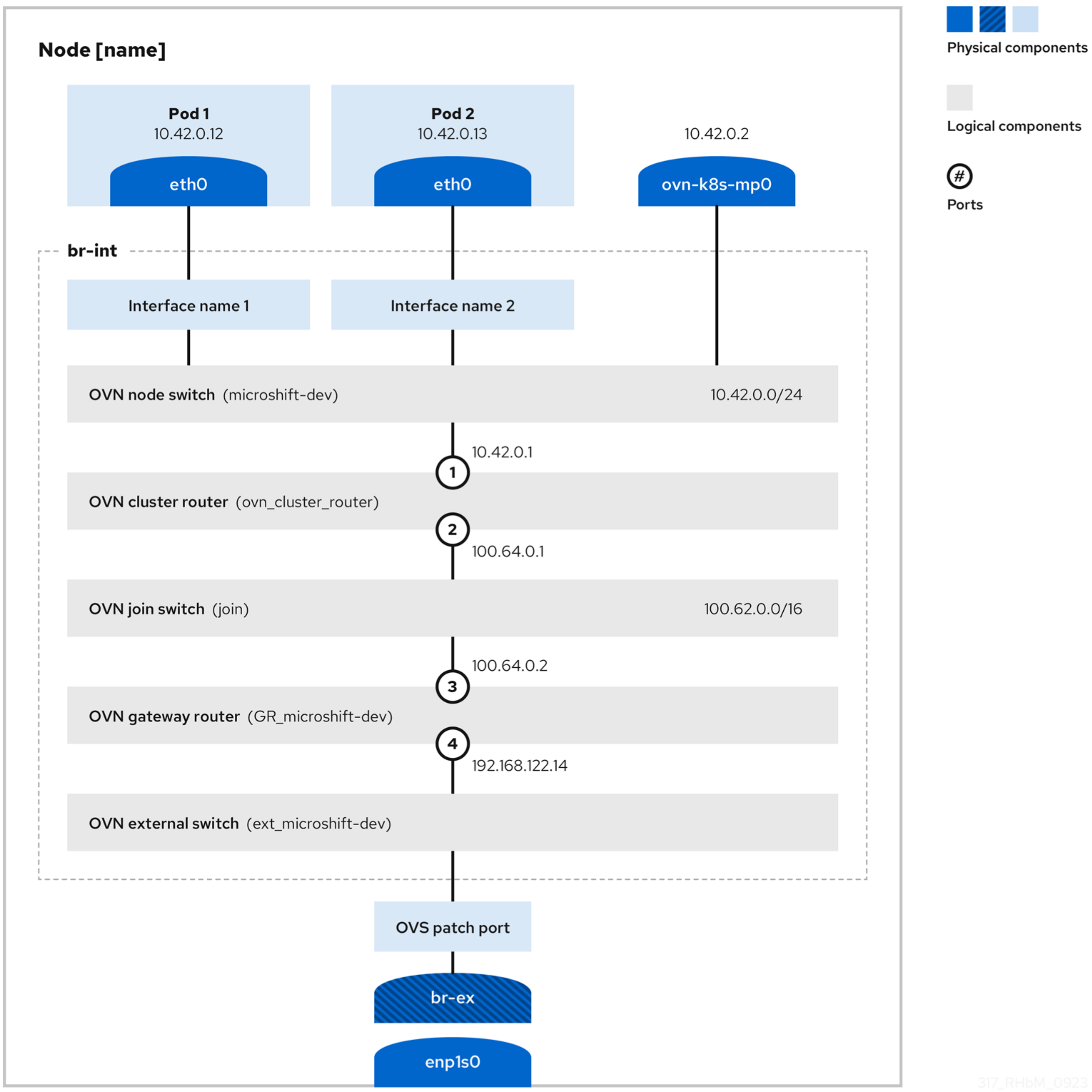
1.7. NETWORK TOPOLOGY

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 (Generic Network Virtualization Encapsulation) tunnel protocol. The pod maximum transmission unit (MTU) for the Geneve tunnel is set to the default route MTU if it is not configured.

To configure the MTU, you must set an equal-to or less-than value than the MTU of the physical interface on the host. A less-than value for the MTU makes room for the required information that is added to the tunnel header before it is transmitted.

OVS runs as a systemd service on the 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.

Red Hat build of MicroShift network topology



1.7.1. Description of the OVN logical components of the virtualized network

OVN node switch

A virtual switch named **<node-name>**. The OVN node switch is named according to the hostname of the node.

- In this example, the **node-name** is **microshift-dev**.

OVN cluster router

A virtual router named **ovn_cluster_router**, also known as the distributed router.

- In this example, the cluster network is **10.42.0.0/16**.

OVN join switch

A virtual switch named **join**.

OVN gateway router

A virtual router named **GR_<node-name>**, also known as the external gateway router.

OVN external switch

A virtual switch named **ext_<node-name>.**

1.7.2. Description of the connections in the network topology figure

- The north-south traffic between the network service and the OVN external switch **ext_microshift-dev** is provided through the host kernel by the gateway bridge **br-ex**.
- The OVN gateway router **GR_microshift-dev** is connected to the external network switch **ext_microshift-dev** through the logical router port 4. Port 4 is attached with the node IP address 192.168.122.14.
- The join switch **join** connects the OVN gateway router **GR_microshift-dev** to the OVN cluster router **ovn_cluster_router**. The IP address range is 100.62.0.0/16.
  - The OVN gateway router **GR_microshift-dev** connects to the OVN join switch **join** through the logical router port 3. Port 3 attaches with the internal IP address 100.64.0.2.
  - The OVN cluster router **ovn_cluster_router** connects to the join switch **join** through the logical router port 2. Port 2 attaches with the internal IP address 100.64.0.1.
- The OVN cluster router **ovn_cluster_router** connects to the node switch **microshift-dev** through the logical router port 1. Port 1 is attached with the OVN cluster network IP address 10.42.0.1.
- The east-west traffic between the pods and the network service is provided by the OVN cluster router **ovn_cluster_router** and the node switch **microshift-dev**. The IP address range is 10.42.0.0/24.
- The east-west traffic between pods is provided by the node switch **microshift-dev** without network address translation (NAT).
- The north-south traffic between the pods and the external network is provided by the OVN cluster router **ovn_cluster_router** and the host network. This router is connected through the **ovn-kubernetes** management port **ovn-k8s-mp0**, with the IP address 10.42.0.2.
- All the pods are connected to the OVN node switch through their interfaces.
  - In this example, Pod 1 and Pod 2 are connected to the node switch through **Interface 1** and **Interface 2**.

CHAPTER 2. UNDERSTANDING NETWORKING SETTINGS

Learn how to apply networking customization and default settings to MicroShift deployments. Each node is contained to a single machine and single 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.

NOTE

To troubleshoot connection problems with the NodePort service, read about the known issue in the Release Notes.

2.1. CREATING AN OVN-KUBERNETES CONFIGURATION FILE

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
```

Example YAML file with default maximum transmission unit (MTU) value

```
mtu: 1400
```

3. To customize your configuration, you can change the MTU value. The table that follows provides details:

Table 2.1. Supported optional OVN-Kubernetes configurations for MicroShift

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

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

```
mtu: 1300
```

2.2. 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.

2.3. DEPLOYING MICROSHIFT BEHIND AN HTTP OR HTTPS PROXY

Deploy a MicroShift cluster behind an HTTP or HTTPS 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 or HTTPS requests when deploying 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 MicroShift.

2.4. USING THE RPM-OSTREE HTTP OR HTTPS PROXY

To use the HTTP or HTTPS proxy in RPM-OStree, you must add a **Service** section to the configuration file and set the **http_proxy environment** variable for the **rpm-ostreed** service.

Procedure

1. Add this setting to the **/etc/systemd/system/rpm-ostreed.service.d/00-proxy.conf** file:

```
[Service]
Environment="http_proxy=http://$PROXY_USER:$PROXY_PASSWORD@$PROXY_SERVE
R:$PROXY_PORT/"
```

2. Next, reload the configuration settings and restart the service to apply your changes.

   a. Reload the configuration settings by running the following command:

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

   b. Restart the **rpm-ostreed** service by running the following command:

   ```
   $ sudo systemctl restart rpm-ostreed.service
   ```

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

To use an HTTP or HTTPS proxy in **CRI-O**, you must add a **Service** section to the configuration file and 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. Create the directory for the configuration file if it does not exist:

```
$ sudo mkdir /etc/systemd/system/crio.service.d/
```

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

```
[Service]
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/"
```

IMPORTANT

You must define the **Service** section of the configuration file for the environment variables or the proxy settings fail to apply.

3. Reload the configuration settings:

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

4. Restart the CRI-O service:

```
$ sudo systemctl restart crio
```

5. Restart the MicroShift service to apply the settings:

```
$ sudo systemctl restart microshift
```

Verification

1. Verify that pods are started by running the following command and examining the output:

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

2. Verify that MicroShift is able to pull container images by running the following command and examining the output:

```
$ sudo crictl images
```

2.6. 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 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 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. 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**.

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. The pod interface **eebee1ce5568761** is named with the first 15 bits of the pod sandbox ID and is plugged into the **br-int** bridge.

4. The pod interface **b47b1995ada84f4** is named with the first 15 bits of the pod sandbox ID and is plugged into the **br-int** bridge.

5. The pod interface **3031f43d67c167f** is named with the first 15 bits of the pod sandbox ID and is plugged into the **br-int** bridge.

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

2.7. DEPLOYING A LOAD BALANCER FOR A WORKLOAD

MicroShift has a built-in implementation of network load balancers. The following example procedure uses the node IP address as the external IP address for the **LoadBalancer** service configuration file. You can use this example as guidance for how to deploy load balancers for your workloads.

Prerequisites

- The OpenShift CLI (**oc**) is installed.
- You have access to the cluster as a user with the cluster administration role.
- You installed a cluster on an infrastructure configured with the OVN-Kubernetes network plugin.

- The **KUBECONFIG** environment variable is set.

Procedure

1. Verify that your pods are running by running the following command:

```
$ oc get pods -A
```

2. Create the example namespace by running the following commands:

```
$ NAMESPACE=nginx-lb-test
```

```
$ oc create ns $NAMESPACE
```

3. The following example deploys three replicas of the test **nginx** application in your namespace:

```
$ oc apply -n $NAMESPACE -f - <<EOF
apiVersion: v1
kind: ConfigMap
metadata:
  name: nginx
data:
  headers.conf: |
    add_header X-Server-IP  \$server_addr always;
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - image: quay.io/packit/nginx-unprivileged
        imagePullPolicy: Always
        name: nginx
        ports:
        - containerPort: 8080
        volumeMounts:
        - name: nginx-configs
          subPath: headers.conf
          mountPath: /etc/nginx/conf.d/headers.conf
        securityContext:
          allowPrivilegeEscalation: false
          seccompProfile:
            type: RuntimeDefault
          capabilities:
            drop: ["ALL"]
          runAsNonRoot: true
      volumes:
        - name: nginx-configs
          configMap:
            name: nginx
            items:
              - key: headers.conf
                path: headers.conf
EOF
```

4. You can verify that the three sample replicas started successfully by running the following command:

```
$ oc get pods -n $NAMESPACE
```

5. Create a **LoadBalancer** service for the **nginx** test application with the following sample commands:

```
$ oc create -n $NAMESPACE -f - <<EOF
apiVersion: v1
kind: Service
metadata:
  name: nginx
spec:
  ports:
  - port: 81
    targetPort: 8080
  selector:
    app: nginx
  type: LoadBalancer
EOF
```

NOTE

You must ensure that the **port** parameter is a host port that is not occupied by other **LoadBalancer** services or Red Hat build of MicroShift components.

6. Verify that the service file exists, that the external IP address is properly assigned, and that the external IP is identical to the node IP by running the following command:

```
$ oc get svc -n $NAMESPACE
```

Example output

```
NAME    TYPE           CLUSTER-IP      EXTERNAL-IP     PORT(S)        AGE
nginx   LoadBalancer   10.43.183.104   192.168.1.241   81:32434/TCP   2m
```

Verification

- The following command forms five connections to the example **nginx** application using the external IP address of the **LoadBalancer** service configuration. The result of the command is a list of those server IP addresses. Verify that the load balancer sends requests to all the running applications with the following command:

```
EXTERNAL_IP=192.168.1.241
seq 5 | xargs -Iz curl -s -I http://$EXTERNAL_IP:81 | grep X-Server-IP
```

The output of the previous command contains different IP addresses if the load balancer is successfully distributing the traffic to the applications, for example:

Example output

```
X-Server-IP: 10.42.0.41
X-Server-IP: 10.42.0.41
X-Server-IP: 10.42.0.43
X-Server-IP: 10.42.0.41
X-Server-IP: 10.42.0.43
```

2.8. BLOCKING EXTERNAL ACCESS TO THE NODEPORT SERVICE ON A SPECIFIC HOST INTERFACE

OVN-Kubernetes does not restrict the host interface where a NodePort service can be accessed from outside a Red Hat build of MicroShift node. The following procedure explains how to block the NodePort service on a specific host interface and restrict external access.

Prerequisites

- You must have an account with root privileges.

Procedure

1. Change the **NODEPORT** variable to the host port number assigned to your Kubernetes NodePort service by running the following command:

```
# export NODEPORT=30700
```

2. Change the **INTERFACE_IP** value to the IP address from the host interface that you want to block. For example:

```
# export INTERFACE_IP=192.168.150.33
```

3. Insert a new rule in the **nat** table PREROUTING chain to drop all packets that match the destination port and IP address. For example:

```
$ sudo nft -a insert rule ip nat PREROUTING tcp dport $NODEPORT ip daddr
$INTERFACE_IP drop
```

4. List the new rule by running the following command:

```
$ sudo nft -a list chain ip nat PREROUTING
table ip nat {
 chain PREROUTING { # handle 1
  type nat hook prerouting priority dstnat; policy accept;
  tcp dport 30700 ip daddr 192.168.150.33 drop # handle 134
  counter packets 108 bytes 18074 jump OVN-KUBE-ETP # handle 116
  counter packets 108 bytes 18074 jump OVN-KUBE-EXTERNALIP # handle 114
  counter packets 108 bytes 18074 jump OVN-KUBE-NODEPORT # handle 112
 }
}
```

NOTE

Note the **handle** number of the newly added rule. You need to remove the **handle** number in the following step.

5. Remove the custom rule with the following sample command:

```
$ sudo nft -a delete rule ip nat PREROUTING handle 134
```

2.9. THE MULTICAST DNS PROTOCOL

You can use the multicast DNS protocol (mDNS) to allow name resolution and service discovery within a Local Area Network (LAN) using multicast exposed on the **5353/UDP** port.

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

2.10. AUDITING EXPOSED NETWORK PORTS

On MicroShift, the host port can be opened by a workload in the following cases. You can check logs to view the network services.

2.10.1. hostNetwork

When a pod is configured with the **hostNetwork:true** setting, the pod is running in the host network namespace. This configuration can independently open host ports. MicroShift component logs cannot be used to track this case, the ports are subject to firewalld rules. If the port opens in firewalld, you can view the port opening in the firewalld debug log.

Prerequisites

- You have root user access to your build host.

Procedure

1. Optional: You can check that the **hostNetwork:true** parameter is set in your ovnkube-node pod by using the following example command:

```
$ sudo oc get pod -n openshift-ovn-kubernetes <ovnkube-node-pod-name> -o json | jq -r
'.spec.hostNetwork' true
```

2. Enable debug in the firewalld log by running the following command:

```
$ sudo vi /etc/sysconfig/firewalld
FIREWALLD_ARGS=--debug=10
```

3. Restart the firewalld service:

```
$ sudo systemctl restart firewalld.service
```

4. To verify that the debug option was added properly, run the following command:

```
$ sudo systemd-cgls -u firewalld.service
```

The firewalld debug log is stored in the **/var/log/firewalld** path.

Example logs for when the port open rule is added:

```
2023-06-28 10:46:37 DEBUG1: config.getZoneByName('public')
2023-06-28 10:46:37 DEBUG1: config.zone.7.addPort('8080', 'tcp')
2023-06-28 10:46:37 DEBUG1: config.zone.7.getSettings()
2023-06-28 10:46:37 DEBUG1: config.zone.7.update('...')
2023-06-28 10:46:37 DEBUG1: config.zone.7.Updated('public')
```

Example logs for when the port open rule is removed:

```
2023-06-28 10:47:57 DEBUG1: config.getZoneByName('public')
2023-06-28 10:47:57 DEBUG2: config.zone.7.Introspect()
2023-06-28 10:47:57 DEBUG1: config.zone.7.removePort('8080', 'tcp')
2023-06-28 10:47:57 DEBUG1: config.zone.7.getSettings()
2023-06-28 10:47:57 DEBUG1: config.zone.7.update('...')
2023-06-28 10:47:57 DEBUG1: config.zone.7.Updated('public')
```

2.10.2. hostPort

You can access the hostPort setting logs in MicroShift. The following logs are examples for the hostPort setting:

Procedure

- You can access the logs by running the following command:

```
$ journalctl -u crio | grep "local port"
```

Example CRI-O logs when the host port is opened:

```
$ Jun 25 16:27:37 rhel92 crio[77216]: time="2023-06-25 16:27:37.033003098+08:00"
level=info msg="Opened local port tcp:443"
```

Example CRI-O logs when the host port is closed:

```
$ Jun 25 16:24:11 rhel92 crio[77216]: time="2023-06-25 16:24:11.342088450+08:00"
level=info msg="Closing host port tcp:443"
```

2.10.3. NodePort and LoadBalancer service

OVN-Kubernetes opens host ports for **NodePort** and **LoadBalancer** service types. These services add iptables rules that take the ingress traffic from the host port and forwards it to the clusterIP. Logs for the **NodePort** and **LoadBalancer** services are presented in the following examples:

Procedure

1. To access the name of your **ovnkube-master** pods, run the following command:

```
$ oc get pods -n openshift-ovn-kubernetes | awk '/ovnkube-master/{print $1}'
```

Example ovnkube-master pod name

```
ovnkube-master-n2shv
```

2. You can access the **NodePort** and **LoadBalancer** services logs using your **ovnkube-master** pod and running the following example command:

```
$ oc logs -n openshift-ovn-kubernetes <ovnkube-master-pod-name> ovnkube-master | grep -
E "OVN-KUBE-NODEPORT|OVN-KUBE-EXTERNALIP"
```

NodePort service:

Example logs in the ovnkube-master container of the ovnkube-master pod when a host port is open:

```
$ I0625 09:07:00.992980 2118395 iptables.go:27] Adding rule in table: nat, chain: OVN-
KUBE-NODEPORT with args: "-p TCP -m addrtype --dst-type LOCAL --dport 32718 -j DNAT
--to-destination 10.96.178.142:8081" for protocol: 0
```

Example logs in the ovnkube-master container of the ovnkube-master pod when a host port is closed:

```
$ Deleting rule in table: nat, chain: OVN-KUBE-NODEPORT with args: "-p TCP -m addrtype -
-dst-type LOCAL --dport 32718 -j DNAT --to-destination 10.96.178.142:8081" for protocol: 0
```

LoadBalancer service:

Example logs in the ovnkube-master container of the ovnkube-master pod when a host port is open:

```
$ I0625 09:34:10.406067  128902 iptables.go:27] Adding rule in table: nat, chain: OVN-
KUBE-EXTERNALIP with args: "-p TCP -d 172.16.47.129 --dport 8081 -j DNAT --to-
destination 10.43.114.94:8081" for protocol: 0
```

Example logs in the ovnkube-master container of the ovnkube-master pod when a host port is closed:

```
$ I0625 09:37:00.976953  128902 iptables.go:63] Deleting rule in table: nat, chain: OVN-
KUBE-EXTERNALIP with args: "-p TCP -d 172.16.47.129 --dport 8081 -j DNAT --to-
destination 10.43.114.94:8081" for protocol: 0
```

CHAPTER 3. NETWORK POLICIES

3.1. ABOUT NETWORK POLICIES

Learn how network policies work for MicroShift to restrict or allow network traffic to pods in your cluster.

3.1.1. How network policy works in MicroShift

In a cluster using the default OVN-Kubernetes Container Network Interface (CNI) plugin for MicroShift, network isolation is controlled by both firewalld, which is configured on the host, and by **NetworkPolicy** objects created within MicroShift. Simultaneous use of firewalld and **NetworkPolicy** is supported.

- Network policies work only within boundaries of OVN-Kubernetes-controlled traffic, so they can apply to every situation except for **hostPort/hostNetwork** enabled pods.
- Firewalld settings also do not apply to **hostPort/hostNetwork** enabled pods.

NOTE

Firewalld rules run before any **NetworkPolicy** is enforced.

WARNING

Network policy does not apply to the host network namespace. Pods with host networking enabled are unaffected by network policy rules. However, pods connecting to the host-networked pods might be affected by the network policy rules.

Network policies cannot block traffic from localhost.

By default, all pods in a MicroShift node are accessible from other pods and network endpoints. To isolate one or more pods in a cluster, you can create **NetworkPolicy** objects to indicate allowed incoming connections. You can create and delete **NetworkPolicy** objects.

If a pod is matched by selectors in one or more **NetworkPolicy** objects, then the pod accepts only connections that are allowed by at least one of those **NetworkPolicy** objects. A pod that is not selected by any **NetworkPolicy** objects is fully accessible.

A network policy applies to only the TCP, UDP, ICMP, and SCTP protocols. Other protocols are not affected.

The following example **NetworkPolicy** objects demonstrate supporting different scenarios:

- Deny all traffic:
  To make a project deny by default, add a **NetworkPolicy** object that matches all pods but accepts no traffic:

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
```

```
metadata:
  name: deny-by-default
spec:
  podSelector: {}
  ingress: []
```

- Allow connections from the default router, which is the ingress in MicroShift:
  To allow connections from the MicroShift default router, add the following **NetworkPolicy** object:

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-openshift-ingress
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          ingresscontroller.operator.openshift.io/deployment-ingresscontroller: default
  podSelector: {}
  policyTypes:
  - Ingress
```

- Only accept connections from pods within the same namespace:
  To make pods accept connections from other pods in the same namespace, but reject all other connections from pods in other namespaces, add the following **NetworkPolicy** object:

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-same-namespace
spec:
  podSelector: {}
  ingress:
  - from:
    - podSelector: {}
```

- Only allow HTTP and HTTPS traffic based on pod labels:
  To enable only HTTP and HTTPS access to the pods with a specific label (**role=frontend** in following example), add a **NetworkPolicy** object similar to the following:

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-http-and-https
spec:
  podSelector:
    matchLabels:
      role: frontend
  ingress:
  - ports:
    - protocol: TCP
```

```
      port: 80
    - protocol: TCP
      port: 443
```

- Accept connections by using both namespace and pod selectors:
  To match network traffic by combining namespace and pod selectors, you can use a **NetworkPolicy** object similar to the following:

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-pod-and-namespace-both
spec:
  podSelector:
    matchLabels:
      name: test-pods
  ingress:
    - from:
      - namespaceSelector:
          matchLabels:
            project: project_name
        podSelector:
          matchLabels:
            name: test-pods
```

NetworkPolicy objects are additive, which means you can combine multiple **NetworkPolicy** objects together to satisfy complex network requirements.

For example, for the **NetworkPolicy** objects defined in previous examples, you can define both **allow-same-namespace** and **allow-http-and-https** policies. That configuration allows the pods with the label **role=frontend** to accept any connection allowed by each policy. That is, connections on any port from pods in the same namespace, and connections on ports **80** and **443** from pods in any namespace.

3.1.2. Optimizations for network policy with OVN-Kubernetes network plugin

When designing your network policy, refer to the following guidelines:

- For network policies with the same **spec.podSelector** spec, it is more efficient to use one network policy with multiple **ingress** or **egress** rules, than multiple network policies with subsets of **ingress** or **egress** rules.

- Every **ingress** or **egress** rule based on the **podSelector** or **namespaceSelector** spec generates the number of OVS flows proportional to **number of pods selected by network policy + number of pods selected by ingress or egress rule**. Therefore, it is preferable to use the **podSelector** or **namespaceSelector** spec that can select as many pods as you need in one rule, instead of creating individual rules for every pod.
  For example, the following policy contains two rules:

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy
spec:
  podSelector: {}
  ingress:
```

```
  - from:
    - podSelector:
        matchLabels:
          role: frontend
  - from:
    - podSelector:
        matchLabels:
          role: backend
```

The following policy expresses those same two rules as one:

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy
spec:
  podSelector: {}
  ingress:
  - from:
    - podSelector:
        matchExpressions:
        - {key: role, operator: In, values: [frontend, backend]}
```

The same guideline applies to the **spec.podSelector** spec. If you have the same **ingress** or **egress** rules for different network policies, it might be more efficient to create one network policy with a common **spec.podSelector** spec. For example, the following two policies have different rules:

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: policy1
spec:
  podSelector:
    matchLabels:
      role: db
  ingress:
  - from:
    - podSelector:
        matchLabels:
          role: frontend
---
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: policy2
spec:
  podSelector:
    matchLabels:
      role: client
  ingress:
  - from:
    - podSelector:
        matchLabels:
          role: frontend
```

The following network policy expresses those same two rules as one:

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: policy3
spec:
  podSelector:
    matchExpressions:
    - {key: role, operator: In, values: [db, client]}
  ingress:
  - from:
    - podSelector:
        matchLabels:
          role: frontend
```

You can apply this optimization when only multiple selectors are expressed as one. In cases where selectors are based on different labels, it may not be possible to apply this optimization. In those cases, consider applying some new labels for network policy optimization specifically.

3.2. CREATING NETWORK POLICIES

You can create a network policy for a namespace.

3.2.1. Example NetworkPolicy object

The following annotates an example NetworkPolicy object:

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-27107 1
spec:
  podSelector: 2
    matchLabels:
      app: mongodb
  ingress:
  - from:
    - podSelector: 3
        matchLabels:
          app: app
    ports: 4
    - protocol: TCP
      port: 27017
```

1. The name of the NetworkPolicy object.
2. A selector that describes the pods to which the policy applies.
3. A selector that matches the pods from which the policy object allows ingress traffic. The selector matches pods in the same namespace as the NetworkPolicy.
4. A list of one or more destination ports on which to accept traffic.

3.2.2. Creating a network policy using the CLI

To define granular rules describing ingress or egress network traffic allowed for namespaces in your cluster, you can create a network policy.

Prerequisites

- You installed the OpenShift CLI (**oc**).
- You are working in the namespace that the network policy applies to.

Procedure

1. Create a policy rule:
   a. Create a **<policy_name>.yaml** file:

      ```
      $ touch <policy_name>.yaml
      ```

      where:

      <policy_name>
      Specifies the network policy file name.

   b. Define a network policy in the file that you just created, such as in the following examples:

      Deny ingress from all pods in all namespaces

      This is a fundamental policy, blocking all cross-pod networking other than cross-pod traffic allowed by the configuration of other Network Policies.

      ```
      kind: NetworkPolicy
      apiVersion: networking.k8s.io/v1
      metadata:
        name: deny-by-default
      spec:
        podSelector: {}
        policyTypes:
        - Ingress
        ingress: []
      ```

      Allow ingress from all pods in the same namespace

      ```
      kind: NetworkPolicy
      apiVersion: networking.k8s.io/v1
      metadata:
        name: allow-same-namespace
      spec:
        podSelector:
        ingress:
        - from:
          - podSelector: {}
      ```

Allow ingress traffic to one pod from a particular namespace

This policy allows traffic to pods labelled **pod-a** from pods running in **namespace-y**.

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-traffic-pod
spec:
  podSelector:
   matchLabels:
      pod: pod-a
  policyTypes:
  - Ingress
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
           kubernetes.io/metadata.name: namespace-y
```

2. To create the network policy object, enter the following command:

```
$ oc apply -f <policy_name>.yaml -n <namespace>
```

where:

<policy_name>
Specifies the network policy file name.

<namespace>
Optional: Specifies the namespace if the object is defined in a different namespace than the current namespace.

Example output

```
networkpolicy.networking.k8s.io/deny-by-default created
```

3.2.3. Creating a default deny all network policy

This is a fundamental policy, blocking all cross-pod networking other than network traffic allowed by the configuration of other deployed network policies. This procedure enforces a default **deny-by-default** policy.

Prerequisites

- You installed the OpenShift CLI (**oc**).
- You are working in the namespace that the network policy applies to.

Procedure

1. Create the following YAML that defines a **deny-by-default** policy to deny ingress from all pods in all namespaces. Save the YAML in the **deny-by-default.yaml** file:

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: deny-by-default
  namespace: default (1)
spec:
  podSelector: {} (2)
  ingress: [] (3)
```

(1) **namespace: default** deploys this policy to the **default** namespace.

(2) **podSelector:** is empty, this means it matches all the pods. Therefore, the policy applies to all pods in the default namespace.

(3) There are no **ingress** rules specified. This causes incoming traffic to be dropped to all pods.

2. Apply the policy by entering the following command:

```
$ oc apply -f deny-by-default.yaml
```

Example output

```
networkpolicy.networking.k8s.io/deny-by-default created
```

3.2.4. Creating a network policy to allow traffic from external clients

With the **deny-by-default** policy in place you can proceed to configure a policy that allows traffic from external clients to a pod with the label **app=web**.

NOTE

Firewalld rules run before any **NetworkPolicy** is enforced.

Follow this procedure to configure a policy that allows external service from the public Internet directly or by using a Load Balancer to access the pod. Traffic is only allowed to a pod with the label **app=web**.

Prerequisites

- You installed the OpenShift CLI (**oc**).
- You are working in the namespace that the network policy applies to.

Procedure

1. Create a policy that allows traffic from the public Internet directly or by using a load balancer to access the pod. Save the YAML in the **web-allow-external.yaml** file:

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: web-allow-external
  namespace: default
spec:
  policyTypes:
  - Ingress
  podSelector:
    matchLabels:
      app: web
  ingress:
    - {}
```

2. Apply the policy by entering the following command:

```
$ oc apply -f web-allow-external.yaml
```

Example output

```
networkpolicy.networking.k8s.io/web-allow-external created
```

3.2.5. Creating a network policy allowing traffic to an application from all namespaces

Follow this procedure to configure a policy that allows traffic from all pods in all namespaces to a particular application.

Prerequisites

- You installed the OpenShift CLI (**oc**).
- You are working in the namespace that the network policy applies to.

Procedure

1. Create a policy that allows traffic from all pods in all namespaces to a particular application. Save the YAML in the **web-allow-all-namespaces.yaml** file:

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: web-allow-all-namespaces
  namespace: default
spec:
  podSelector:
    matchLabels:
      app: web 1
  policyTypes:
  - Ingress
  ingress:
  - from:
    - namespaceSelector: {} 2
```

1 Applies the policy only to **app:web** pods in default namespace.

2 Selects all pods in all namespaces.

NOTE

By default, if you omit specifying a **namespaceSelector** it does not select any namespaces, which means the policy allows traffic only from the namespace the network policy is deployed to.

2. Apply the policy by entering the following command:

```
$ oc apply -f web-allow-all-namespaces.yaml
```

Example output

```
networkpolicy.networking.k8s.io/web-allow-all-namespaces created
```

Verification

1. Start a web service in the **default** namespace by entering the following command:

```
$ oc run web --namespace=default --image=nginx --labels="app=web" --expose --port=80
```

2. Run the following command to deploy an **alpine** image in the **secondary** namespace and to start a shell:

```
$ oc run test-$RANDOM --namespace=secondary --rm -i -t --image=alpine -- sh
```

3. Run the following command in the shell and observe that the request is allowed:

```
# wget -qO- --timeout=2 http://web.default
```

Expected output

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
```

```
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

3.2.6. Creating a network policy allowing traffic to an application from a namespace

Follow this procedure to configure a policy that allows traffic to a pod with the label **app=web** from a particular namespace. You might want to do this to:

- Restrict traffic to a production database only to namespaces where production workloads are deployed.
- Enable monitoring tools deployed to a particular namespace to scrape metrics from the current namespace.

Prerequisites

- You installed the OpenShift CLI (**oc**).
- You are working in the namespace that the network policy applies to.

Procedure

1. Create a policy that allows traffic from all pods in a particular namespaces with a label **purpose=production**. Save the YAML in the **web-allow-prod.yaml** file:

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: web-allow-prod
  namespace: default
spec:
  podSelector:
    matchLabels:
      app: web (1)
  policyTypes:
  - Ingress
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          purpose: production (2)
```

(1) Applies the policy only to **app:web** pods in the default namespace.

(2) Restricts traffic to only pods in namespaces that have the label **purpose=production**.

2. Apply the policy by entering the following command:

```
$ oc apply -f web-allow-prod.yaml
```

Example output

```
networkpolicy.networking.k8s.io/web-allow-prod created
```

Verification

1. Start a web service in the **default** namespace by entering the following command:

```
$ oc run web --namespace=default --image=nginx --labels="app=web" --expose --port=80
```

2. Run the following command to create the **prod** namespace:

```
$ oc create namespace prod
```

3. Run the following command to label the **prod** namespace:

```
$ oc label namespace/prod purpose=production
```

4. Run the following command to create the **dev** namespace:

```
$ oc create namespace dev
```

5. Run the following command to label the **dev** namespace:

```
$ oc label namespace/dev purpose=testing
```

6. Run the following command to deploy an **alpine** image in the **dev** namespace and to start a shell:

```
$ oc run test-$RANDOM --namespace=dev --rm -i -t --image=alpine -- sh
```

7. Run the following command in the shell and observe that the request is blocked:

```
# wget -qO- --timeout=2 http://web.default
```

Expected output

```
wget: download timed out
```

8. Run the following command to deploy an **alpine** image in the **prod** namespace and start a shell:

```
$ oc run test-$RANDOM --namespace=prod --rm -i -t --image=alpine -- sh
```

9. Run the following command in the shell and observe that the request is allowed:

```
# wget -qO- --timeout=2 http://web.default
```

Expected output

```
<!DOCTYPE html>
```

```
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

3.3. EDITING A NETWORK POLICY

You can edit an existing network policy for a namespace. Typical edits might include changes to the pods to which the policy applies, allowed ingress traffic, and the destination ports on which to accept traffic. The **apiVersion**, **kind**, and **name** fields must not be changed when editing **NetworkPolicy** objects, as these define the resource itself.

3.3.1. Editing a network policy

You can edit a network policy in a namespace.

Prerequisites

- You installed the OpenShift CLI (**oc**).
- You are working in the namespace where the network policy exists.

Procedure

1. Optional: To list the network policy objects in a namespace, enter the following command:

   ```
   $ oc get networkpolicy
   ```

   where:

   <namespace>

   Optional: Specifies the namespace if the object is defined in a different namespace than the current namespace.

2. Edit the network policy object.

- If you saved the network policy definition in a file, edit the file and make any necessary changes, and then enter the following command.

  ```
  $ oc apply -n <namespace> -f <policy_file>.yaml
  ```

  where:

  <namespace>
  Optional: Specifies the namespace if the object is defined in a different namespace than the current namespace.

  <policy_file>
  Specifies the name of the file containing the network policy.

- If you need to update the network policy object directly, enter the following command:

  ```
  $ oc edit networkpolicy <policy_name> -n <namespace>
  ```

  where:

  <policy_name>
  Specifies the name of the network policy.

  <namespace>
  Optional: Specifies the namespace if the object is defined in a different namespace than the current namespace.

3. Confirm that the network policy object is updated.

   ```
   $ oc describe networkpolicy <policy_name> -n <namespace>
   ```

   where:

   <policy_name>
   Specifies the name of the network policy.

   <namespace>
   Optional: Specifies the namespace if the object is defined in a different namespace than the current namespace.

3.3.2. Example NetworkPolicy object

The following annotates an example NetworkPolicy object:

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-27107 1
spec:
  podSelector: 2
    matchLabels:
      app: mongodb
  ingress:
  - from:
    - podSelector: 3
```

```
      matchLabels:
        app: app
  ports: (4)
  - protocol: TCP
    port: 27017
```

(1) The name of the NetworkPolicy object.

(2) A selector that describes the pods to which the policy applies.

(3) A selector that matches the pods from which the policy object allows ingress traffic. The selector matches pods in the same namespace as the NetworkPolicy.

(4) A list of one or more destination ports on which to accept traffic.

3.4. DELETING A NETWORK POLICY

You can delete a network policy from a namespace.

3.4.1. Deleting a network policy using the CLI

You can delete a network policy in a namespace.

Prerequisites

- You installed the OpenShift CLI (**oc**).
- You are working in the namespace where the network policy exists.

Procedure

- To delete a network policy object, enter the following command:

  ```
  $ oc delete networkpolicy <policy_name> -n <namespace>
  ```

  where:

  <policy_name>
  Specifies the name of the network policy.

  <namespace>
  Optional: Specifies the namespace if the object is defined in a different namespace than the current namespace.

  Example output

  ```
  networkpolicy.networking.k8s.io/default-deny deleted
  ```

3.5. VIEWING A NETWORK POLICY

Use the following procedure to view a network policy for a namespace.

3.5.1. Viewing network policies using the CLI

You can examine the network policies in a namespace.

Prerequisites

- You installed the OpenShift CLI (**oc**).
- You are working in the namespace where the network policy exists.

Procedure

- List network policies in a namespace:
  - To view network policy objects defined in a namespace, enter the following command:

    ```
    $ oc get networkpolicy
    ```

  - Optional: To examine a specific network policy, enter the following command:

    ```
    $ oc describe networkpolicy <policy_name> -n <namespace>
    ```

    where:

    <policy_name>
    Specifies the name of the network policy to inspect.

    <namespace>
    Optional: Specifies the namespace if the object is defined in a different namespace than the current namespace.

    For example:

    ```
    $ oc describe networkpolicy allow-same-namespace
    ```

    Output for oc describe command

    ```
    Name:         allow-same-namespace
    Namespace:    ns1
    Created on:   2021-05-24 22:28:56 -0400 EDT
    Labels:       <none>
    Annotations:  <none>
    Spec:
      PodSelector:     <none> (Allowing the specific traffic to all pods in this namespace)
      Allowing ingress traffic:
        To Port: <any> (traffic allowed to all ports)
        From:
          PodSelector: <none>
      Not affecting egress traffic
      Policy Types: Ingress
    ```

CHAPTER 4. USING A FIREWALL

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

4.1. ABOUT NETWORK TRAFFIC THROUGH THE FIREWALL

Firewalld is a networking service that runs in the background and responds to connection requests, creating a dynamic customizable host-based firewall. If you are using Red Hat Enterprise Linux for Edge (RHEL for Edge) with MicroShift, firewalld should already be installed and you just need to configure it. Details are provided in procedures that follow. Overall, 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 You can configure the **firewalld** service by using the following procedures. In most cases, firewalld is part of RHEL for Edge installations. If you do not have firewalld, you can install it with the simple procedure in this section.

IMPORTANT

MicroShift pods must have access to the internal CoreDNS component and API servers.

Additional resources

- Required firewall settings
- Allowing network traffic through the firewall

4.2. INSTALLING THE FIREWALLD SERVICE

If you are using RHEL for Edge, firewalld should be installed. To use the service, you can simply configure it. The following procedure can be used if you do not have firewalld, but want to use it.

Install and run the **firewalld** service for MicroShift by using the following steps.

Procedure

1. Optional: Check for firewalld on your system by running the following command:

```
$ rpm -q firewalld
```

2. If the **firewalld** service is not installed, run the following command:

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

3. To start the firewall, run the following command:

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

4.3. 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 4.1. Firewall IP address settings

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

4.4. USING OPTIONAL PORT SETTINGS

The 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 4.2. Optional ports

Port(s)	Protocol(s)	Description
80	TCP	HTTP port used to serve applications through the OpenShift Container Platform router.

Port(s)	Protocol(s)	Description
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.
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 MicroShift, such as port 6443 for the API server, for example, ports 80 and 443 for applications exposed through the router.

Example command

- Configuring a port for the MicroShift API server:

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

To close unnecessary ports in your MicroShift instance, follow the procedure in "Closing unused or unnecessary ports to enhance network security".

Additional resources

- Closing unused or unnecessary ports to enhance network security

4.5. ADDING SERVICES TO OPEN PORTS

On a MicroShift instance, you can open services on ports by using the **firewall-cmd** command.

Procedure

1. Optional: You can view all predefined services in firewalld by running the following command

```
$ sudo firewall-cmd --get-services
```

2. To open a service that you want on a default port, run the following example command:

```
$ sudo firewall-cmd --add-service=mdns
```

4.6. ALLOWING NETWORK TRAFFIC THROUGH THE FIREWALL

You can allow network traffic through the firewall by configuring the IP address range and inserting the DNS server to allow internal traffic from pods through the network gateway.

Procedure

1. Use one of the following commands to set the IP address range:
    a. Configure the IP address range with default values by running the following command:

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

    b. 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
```

4.6.1. 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 the settings:

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

4.7. 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
```

4.8. OVERVIEW OF FIREWALL PORTS WHEN A SERVICE IS EXPOSED

Firewalld is often active when you run services on MicroShift. This can disrupt certain services on MicroShift because traffic to the ports might be blocked by the firewall. You must ensure that the necessary firewall ports are open if you want certain services to be accessible from outside the host. There are several options for opening your ports:

- Services of the **NodePort** and **LoadBalancer** type are automatically available with OVN-Kubernetes.
  In these cases, OVN-Kubernetes adds iptables rules so the traffic to the node IP address is delivered to the relevant ports. This is done using the PREROUTING rule chain and is then forwarded to the OVN-K to bypass the firewalld rules for local host ports and services. Iptables and firewalld are backed by nftables in RHEL 9. The nftables rules, which the iptables generates, always have priority over the rules that the firewalld generates.

- Pods with the **HostPort** parameter settings are automatically available. This also includes the **router-default** pod, which uses ports 80 and 443.
  For **HostPort** pods, the CRI-O config sets up iptables DNAT (Destination Network Address Translation) to the pod's IP address and port.

These methods function for clients whether they are on the same host or on a remote host. The iptables rules, which are added by OVN-Kubernetes and CRI-O, attach to the PREROUTING and OUTPUT chains. The local traffic goes through the OUTPUT chain with the interface set to the **lo** type. The DNAT runs before it hits filler rules in the INPUT chain.

Because the MicroShift API server does not run in CRI-O, it is subject to the firewall configurations. You can open port 6443 in the firewall to access the API server in your MicroShift cluster.

4.9. ADDITIONAL RESOURCES

- RHEL: Using and configuring firewalld

- RHEL: Viewing the current status of firewalld

4.10. KNOWN FIREWALL ISSUE

- To avoid breaking traffic flows with a firewall reload or restart, execute firewall commands before starting RHEL. The CNI driver in 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 MicroShift is running, manually restart **ovnkube-master** pod in the **openshift-ovn-kubernetes** namespace to reset the rules controlled by the CNI driver.

CHAPTER 5. CONFIGURING NETWORK SETTINGS FOR FULLY DISCONNECTED HOSTS

Learn how to apply networking customization and settings to run MicroShift on fully disconnected hosts. A disconnected host should be the Red Hat Enterprise Linux (RHEL) operating system, versions 9.0+, whether real or virtual, that runs without network connectivity.

5.1. PREPARING NETWORKING FOR FULLY DISCONNECTED HOSTS

Use the procedure that follows to start and run MicroShift clusters on devices running fully disconnected operating systems. A MicroShift host is considered fully disconnected if it has no external network connectivity.

Typically this means that the device does not have an attached network interface controller (NIC) to provide a subnet. These steps can also be completed on a host with a NIC that is removed after setup. You can also automate these steps on a host that does not have a NIC by using the **%post** phase of a Kickstart file.

IMPORTANT

Configuring networking settings for disconnected environments is necessary because MicroShift requires a network device to support cluster communication. To meet this requirement, you must configure MicroShift networking settings to use the "fake" IP address you assign to the system loopback device during setup.

5.1.1. Procedure summary

To run MicroShift on a disconnected host, the following steps are required:

Prepare the host

- Stop MicroShift if it is currently running and clean up changes the service has made to the network.
- Set a persistent hostname.
- Add a “fake” IP address on the loopback interface.
- Configure DNS to use the fake IP as local name server.
- Add an entry for the hostname to **/etc/hosts**.

Update the MicroShift configuration

- Define the **nodeIP** parameter as the new loopback IP address.
- Set the **.node.hostnameOverride** parameter to the persistent hostname.

For the changes to take effect

- Disable the default NIC if attached.
- Restart the host or device.

After starting, MicroShift runs using the loopback device for within-cluster communication.

5.2. RESTORING MICROSHIFT NETWORKING SETTINGS TO DEFAULT

You can remove networking customizations and return the network to default settings by stopping MicroShift and running a clean-up script.

Prerequisites

- RHEL 9 or newer.
- MicroShift 4.14 or newer.
- Access to the host CLI.

Procedure

1. Stop the MicroShift service by running the following command:

   ```
   $ sudo systemctl stop microshift
   ```

2. Stop the **kubepods.slice** systemd unit by running the following command:

   ```
   $ sudo systemctl stop kubepods.slice
   ```

3. MicroShift installs a helper script to undo network changes made by OVN-K. Run the cleanup script by entering the following command:

   ```
   $ sudo /usr/bin/microshift-cleanup-data --ovn
   ```

5.3. CONFIGURING THE NETWORKING SETTINGS FOR FULLY DISCONNECTED HOSTS

To configure the networking settings for running MicroShift on a fully disconnected host, you must prepare the host, update the networking configuration, then restart to apply the new settings. All commands are executed from the host CLI.

Prerequisites

- RHEL 9 or newer.
- MicroShift 4.14 or newer.
- Access to the host CLI.
- A valid IP address chosen to avoid both internal and potential future external IP conflicts when running MicroShift.
- MicroShift networking settings are set to defaults.

IMPORTANT

The following procedure is for use cases in which access to the MicroShift cluster is not required after devices are deployed in the field. There is no remote cluster access after the network connection is removed.

Procedure

1. Add a fake IP address to the loopback interface by running the following command:

   ```
   $ IP="10.44.0.1" (1)
   $ sudo nmcli con add type loopback con-name stable-microshift ifname lo ip4 ${IP}/32
   ```

   (1) The fake IP address used in this example is “10.44.0.1”.

   NOTE

   Any valid IP works if it avoids both internal MicroShift and potential future external IP conflicts. This can be any subnet that does not collide with the MicroShift node subnet or is be accessed by other services on the device.

2. Configure the DNS interface to use the local name server by setting modifying the settings to ignore automatic DNS and reset it to the local name server:

   a. Bypass the automatic DNS by running the following command:

      ```
      $ sudo nmcli conn modify stable-microshift ipv4.ignore-auto-dns yes
      ```

   b. Point the DNS interface to use the local name server:

      ```
      $ sudo nmcli conn modify stable-microshift ipv4.dns "10.44.1.1"
      ```

3. Get the hostname of the device by running the following command:

   ```
   $ NAME="$(hostnamectl hostname)"
   ```

4. Add an entry for the hostname of the node in the **/etc/hosts** file by running the following command:

   ```
   $ echo "$IP $NAME" | sudo tee -a /etc/hosts >/dev/null
   ```

5. Update the MicroShift configuration file by adding the following YAML snippet to **/etc/microshift/config.yaml**:

   ```
   sudo tee /etc/microshift/config.yaml > /dev/null <<EOF
   node:
     hostnameOverride: hostnameOverride: $(echo $NAME)
     nodeIP: $(echo $IP)
   EOF
   ```

6. MicroShift is now ready to use the loopback device for cluster communications. Finish preparing the device for offline use.

a. If the device currently has a NIC attached, disconnect the device from the network.

b. Shut down the device and disconnect the NIC.

c. Restart the device for the offline configuration to take effect.

7. Restart the MicroShift host to apply the configuration changes by running the following command:

```
$ sudo systemctl reboot (1)
```

(1) This step restarts the cluster. Wait for the greenboot health check to report the system healthy before implementing verification.

Verification

At this point, network access to the MicroShift host has been severed. If you have access to the host terminal, you can use the host CLI to verify that the cluster has started in a stable state.

1. Verify that the MicroShift cluster is running by entering the following command:

```
$ export KUBECONFIG=/var/lib/microshift/resources/kubeadmin/kubeconfig
$ sudo -E oc get pods -A
```

Example output

```
NAMESPACE                  NAME                                   READY   STATUS    RESTARTS
AGE
kube-system                csi-snapshot-controller-74d566564f-66n2f   1/1     Running   0
1m
kube-system                csi-snapshot-webhook-69bdff8879-xs6mb      1/1     Running   0
1m
openshift-dns              dns-default-dxglm                      2/2     Running   0          1m
openshift-dns              node-resolver-dbf5v                    1/1     Running   0          1m
openshift-ingress          router-default-8575d888d8-xmq9p        1/1     Running   0
1m
openshift-ovn-kubernetes   ovnkube-master-gcsx8                   4/4     Running   1          1m
openshift-ovn-kubernetes   ovnkube-node-757mf                     1/1     Running   1          1m
openshift-service-ca       service-ca-7d7c579f54-68jt4            1/1     Running   0          1m
openshift-storage          topolvm-controller-6d777f795b-bx22r    5/5     Running   0
1m
openshift-storage          topolvm-node-fcf8l                     4/4     Running   0          1m
```