OpenShift Container Platform 4.13

Networking

Configuring and managing cluster networking

OpenShift Container Platform 4.13 Networking

Configuring and managing cluster networking

Legal Notice

Abstract

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

Table of Contents

CHAPTER 1. ABOUT NETWORKING

Red Hat OpenShift Networking is an ecosystem of features, plugins and advanced networking capabilities that extend Kubernetes networking with the advanced networking-related features that your cluster needs to manage its network traffic for one or multiple hybrid clusters. This ecosystem of networking capabilities integrates ingress, egress, load balancing, high-performance throughput, security, inter- and intra-cluster traffic management and provides role-based observability tooling to reduce its natural complexities.

The following list highlights some of the most commonly used Red Hat OpenShift Networking features available on your cluster:

- Primary cluster network provided by either of the following Container Network Interface (CNI) plugins:

  - OVN-Kubernetes network plugin, the default plugin

  - OpenShift SDN network plugin

- Certified 3rd-party alternative primary network plugins

- Cluster Network Operator for network plugin management

- Ingress Operator for TLS encrypted web traffic

- DNS Operator for name assignment

- MetalLB Operator for traffic load balancing on bare metal clusters

- IP failover support for high-availability

- Additional hardware network support through multiple CNI plugins, including for macvlan, ipvlan, and SR-IOV hardware networks

- IPv4, IPv6, and dual stack addressing

- Hybrid Linux-Windows host clusters for Windows-based workloads

- Red Hat OpenShift Service Mesh for discovery, load balancing, service-to-service authentication, failure recovery, metrics, and monitoring of services

- Single-node OpenShift

- Network Observability Operator for network debugging and insights

- Submariner and Red Hat Application Interconnect technologies for inter-cluster networking

CHAPTER 2. UNDERSTANDING NETWORKING

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

- Service types, such as node ports or load balancers

- 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 network, but clients outside the cluster do not have networking access. When you expose your application to external traffic, giving each pod its own IP address means that pods can be treated like physical hosts or virtual machines in terms of port allocation, networking, naming, service discovery, load balancing, application configuration, and migration.

> **NOTE**
>
> Some cloud platforms offer metadata APIs that listen on the 169.254.169.254 IP address, a link-local IP address in the IPv4 **169.254.0.0/16** CIDR block.
>
> This CIDR block is not reachable from the pod network. Pods that need access to these IP addresses must be given host network access by setting the **spec.hostNetwork** field in the pod spec to **true**.
>
> If you allow a pod host network access, you grant the pod privileged access to the underlying network infrastructure.

2.1. OPENSHIFT CONTAINER PLATFORM DNS

If you are running multiple services, such as front-end and back-end services for use with multiple pods, environment variables are created for user names, service IPs, and more so the front-end pods can communicate with the back-end services. If the service is deleted and recreated, a new IP address can be assigned to the service, and requires the front-end pods to be recreated to pick up the updated values for the service IP environment variable. Additionally, the back-end service must be created before any of the front-end pods to ensure that the service IP is generated properly, and that it can be provided to the front-end pods as an environment variable.

For this reason, OpenShift Container Platform has a built-in DNS so that the services can be reached by the service DNS as well as the service IP/port.

2.2. OPENSHIFT CONTAINER PLATFORM INGRESS OPERATOR

When you create your OpenShift Container Platform cluster, pods and services running on the cluster are each allocated their own IP addresses. The IP addresses are accessible to other pods and services running nearby but are not accessible to outside clients. The Ingress Operator implements the **IngressController** API and is the component responsible for enabling external access to OpenShift Container Platform cluster services.

The Ingress Operator makes it possible for external clients to access your service by deploying and managing one or more HAProxy-based Ingress Controllers to handle routing. You can use the Ingress Operator to route traffic by specifying OpenShift Container Platform **Route** and Kubernetes **Ingress** resources. Configurations within the Ingress Controller, such as the ability to define **endpointPublishingStrategy** type and internal load balancing, provide ways to publish Ingress Controller endpoints.

2.2.1. Comparing routes and Ingress

The Kubernetes Ingress resource in OpenShift Container Platform implements the Ingress Controller with a shared router service that runs as a pod inside the cluster. The most common way to manage Ingress traffic is with the Ingress Controller. You can scale and replicate this pod like any other regular pod. This router service is based on [HAProxy](#), which is an open source load balancer solution.

The OpenShift Container Platform route provides Ingress traffic to services in the cluster. Routes provide advanced features that might not be supported by standard Kubernetes Ingress Controllers, such as TLS re-encryption, TLS passthrough, and split traffic for blue-green deployments.

Ingress traffic accesses services in the cluster through a route. Routes and Ingress are the main resources for handling Ingress traffic. Ingress provides features similar to a route, such as accepting external requests and delegating them based on the route. However, with Ingress you can only allow certain types of connections: HTTP/2, HTTPS and server name identification (SNI), and TLS with certificate. In OpenShift Container Platform, routes are generated to meet the conditions specified by the Ingress resource.

2.3. GLOSSARY OF COMMON TERMS FOR OPENSHIFT CONTAINER PLATFORM NETWORKING

This glossary defines common terms that are used in the networking content.

authentication

To control access to an OpenShift Container Platform cluster, a cluster administrator can configure user authentication and ensure only approved users access the cluster. To interact with an OpenShift Container Platform cluster, you must authenticate to the OpenShift Container Platform API. You can authenticate by providing an OAuth access token or an X.509 client certificate in your requests to the OpenShift Container Platform API.

AWS Load Balancer Operator

The AWS Load Balancer (ALB) Operator deploys and manages an instance of the **aws-load-balancer-controller**.

Cluster Network Operator

The Cluster Network Operator (CNO) deploys and manages the cluster network components in an OpenShift Container Platform cluster. This includes deployment of the Container Network Interface (CNI) network plugin selected for the cluster during installation.

config map

A config map provides a way to inject configuration data into pods. You can reference the data stored in a config map in a volume of type **ConfigMap**. Applications running in a pod can use this data.

custom resource (CR)

A CR is extension of the Kubernetes API. You can create custom resources.

DNS

Cluster DNS is a DNS server which serves DNS records for Kubernetes services. Containers started by Kubernetes automatically include this DNS server in their DNS searches.

DNS Operator

The DNS Operator deploys and manages CoreDNS to provide a name resolution service to pods. This enables DNS-based Kubernetes Service discovery in OpenShift Container Platform.

deployment

A Kubernetes resource object that maintains the life cycle of an application.

domain

Domain is a DNS name serviced by the Ingress Controller.

egress

The process of data sharing externally through a network's outbound traffic from a pod.

External DNS Operator

The External DNS Operator deploys and manages ExternalDNS to provide the name resolution for services and routes from the external DNS provider to OpenShift Container Platform.

HTTP-based route

An HTTP-based route is an unsecured route that uses the basic HTTP routing protocol and exposes a service on an unsecured application port.

Ingress

The Kubernetes Ingress resource in OpenShift Container Platform implements the Ingress Controller with a shared router service that runs as a pod inside the cluster.

Ingress Controller

The Ingress Operator manages Ingress Controllers. Using an Ingress Controller is the most common way to allow external access to an OpenShift Container Platform cluster.

installer-provisioned infrastructure

The installation program deploys and configures the infrastructure that the cluster runs on.

kubelet

A primary node agent that runs on each node in the cluster to ensure that containers are running in a pod.

Kubernetes NMState Operator

The Kubernetes NMState Operator provides a Kubernetes API for performing state-driven network configuration across the OpenShift Container Platform cluster's nodes with NMState.

kube-proxy

Kube-proxy is a proxy service which runs on each node and helps in making services available to the external host. It helps in forwarding the request to correct containers and is capable of performing primitive load balancing.

load balancers

OpenShift Container Platform uses load balancers for communicating from outside the cluster with services running in the cluster.

MetalLB Operator

As a cluster administrator, you can add the MetalLB Operator to your cluster so that when a service of type **LoadBalancer** is added to the cluster, MetalLB can add an external IP address for the service.

multicast

With IP multicast, data is broadcast to many IP addresses simultaneously.

namespaces

A namespace isolates specific system resources that are visible to all processes. Inside a namespace, only processes that are members of that namespace can see those resources.

networking

Network information of a OpenShift Container Platform cluster.

node

A worker machine in the OpenShift Container Platform cluster. A node is either a virtual machine (VM) or a physical machine.

OpenShift Container Platform Ingress Operator

The Ingress Operator implements the **IngressController** API and is the component responsible for enabling external access to OpenShift Container Platform services.

pod

One or more containers with shared resources, such as volume and IP addresses, running in your OpenShift Container Platform cluster. A pod is the smallest compute unit defined, deployed, and managed.

PTP Operator

The PTP Operator creates and manages the **linuxptp** services.

route

The OpenShift Container Platform route provides Ingress traffic to services in the cluster. Routes provide advanced features that might not be supported by standard Kubernetes Ingress Controllers, such as TLS re-encryption, TLS passthrough, and split traffic for blue-green deployments.

scaling

Increasing or decreasing the resource capacity.

service

Exposes a running application on a set of pods.

Single Root I/O Virtualization (SR-IOV) Network Operator

The Single Root I/O Virtualization (SR-IOV) Network Operator manages the SR-IOV network devices and network attachments in your cluster.

software-defined networking (SDN)

OpenShift Container Platform uses a software-defined networking (SDN) approach to provide a unified cluster network that enables communication between pods across the OpenShift Container Platform cluster.

Stream Control Transmission Protocol (SCTP)

SCTP is a reliable message based protocol that runs on top of an IP network.

taint

Taints and tolerations ensure that pods are scheduled onto appropriate nodes. You can apply one or more taints on a node.

toleration

You can apply tolerations to pods. Tolerations allow the scheduler to schedule pods with matching taints.

web console

A user interface (UI) to manage OpenShift Container Platform.

CHAPTER 3. ACCESSING HOSTS

Learn how to create a bastion host to access OpenShift Container Platform instances and access the control plane nodes with secure shell (SSH) access.

3.1. ACCESSING HOSTS ON AMAZON WEB SERVICES IN AN INSTALLER-PROVISIONED INFRASTRUCTURE CLUSTER

The OpenShift Container Platform installer does not create any public IP addresses for any of the Amazon Elastic Compute Cloud (Amazon EC2) instances that it provisions for your OpenShift Container Platform cluster. To be able to SSH to your OpenShift Container Platform hosts, you must follow this procedure.

Procedure

1. Create a security group that allows SSH access into the virtual private cloud (VPC) created by the **openshift-install** command.

2. Create an Amazon EC2 instance on one of the public subnets the installer created.

3. Associate a public IP address with the Amazon EC2 instance that you created.
   Unlike with the OpenShift Container Platform installation, you should associate the Amazon EC2 instance you created with an SSH keypair. It does not matter what operating system you choose for this instance, as it will simply serve as an SSH bastion to bridge the internet into your OpenShift Container Platform cluster's VPC. The Amazon Machine Image (AMI) you use does matter. With Red Hat Enterprise Linux CoreOS (RHCOS), for example, you can provide keys via Ignition, like the installer does.

4. After you provisioned your Amazon EC2 instance and can SSH into it, you must add the SSH key that you associated with your OpenShift Container Platform installation. This key can be different from the key for the bastion instance, but does not have to be.

   > **NOTE**
   >
   > Direct SSH access is only recommended for disaster recovery. When the Kubernetes API is responsive, run privileged pods instead.

5. Run **oc get nodes**, inspect the output, and choose one of the nodes that is a master. The hostname looks similar to **ip-10-0-1-163.ec2.internal**.

6. From the bastion SSH host you manually deployed into Amazon EC2, SSH into that control plane host. Ensure that you use the same SSH key you specified during the installation:

   ```
   $ ssh -i <ssh-key-path> core@<master-hostname>
   ```

CHAPTER 4. NETWORKING OPERATORS OVERVIEW

OpenShift Container Platform supports multiple types of networking Operators. You can manage the cluster networking using these networking Operators.

4.1. CLUSTER NETWORK OPERATOR

The Cluster Network Operator (CNO) deploys and manages the cluster network components in an OpenShift Container Platform cluster. This includes deployment of the Container Network Interface (CNI) network plugin selected for the cluster during installation. For more information, see Cluster Network Operator in OpenShift Container Platform.

4.2. DNS OPERATOR

The DNS Operator deploys and manages CoreDNS to provide a name resolution service to pods. This enables DNS-based Kubernetes Service discovery in OpenShift Container Platform. For more information, see DNS Operator in OpenShift Container Platform .

4.3. INGRESS OPERATOR

When you create your OpenShift Container Platform cluster, pods and services running on the cluster are each allocated IP addresses. The IP addresses are accessible to other pods and services running nearby but are not accessible to external clients. The Ingress Operator implements the Ingress Controller API and is responsible for enabling external access to OpenShift Container Platform cluster services. For more information, see Ingress Operator in OpenShift Container Platform .

4.4. EXTERNAL DNS OPERATOR

The External DNS Operator deploys and manages ExternalDNS to provide the name resolution for services and routes from the external DNS provider to OpenShift Container Platform. For more information, see Understanding the External DNS Operator .

4.5. INGRESS NODE FIREWALL OPERATOR

The Ingress Node Firewall Operator uses an extended Berkley Packet Filter (eBPF) and eXpress Data Path (XDP) plugin to process node firewall rules, update statistics and generate events for dropped traffic. The operator manages ingress node firewall resources, verifies firewall configuration, does not allow incorrectly configured rules that can prevent cluster access, and loads ingress node firewall XDP programs to the selected interfaces in the rule's object(s). For more information, see Understanding the Ingress Node Firewall Operator

4.6. NETWORK OBSERVABILITY OPERATOR

The Network Observability Operator is an optional Operator that allows cluster administrators to observe the network traffic for OpenShift Container Platform clusters. The Network Observability Operator uses the eBPF technology to create network flows. The network flows are then enriched with OpenShift Container Platform information and stored in Loki. You can view and analyze the stored network flows information in the OpenShift Container Platform console for further insight and troubleshooting. For more information, see About Network Observability Operator.

CHAPTER 5. CLUSTER NETWORK OPERATOR IN OPENSHIFT CONTAINER PLATFORM

The Cluster Network Operator (CNO) deploys and manages the cluster network components on an OpenShift Container Platform cluster, including the Container Network Interface (CNI) network plugin selected for the cluster during installation.

5.1. CLUSTER NETWORK OPERATOR

The Cluster Network Operator implements the **network** API from the **operator.openshift.io** API group. The Operator deploys the OVN-Kubernetes network plugin, or the network provider plugin that you selected during cluster installation, by using a daemon set.

Procedure

The Cluster Network Operator is deployed during installation as a Kubernetes **Deployment**.

1. Run the following command to view the Deployment status:

   ```
   $ oc get -n openshift-network-operator deployment/network-operator
   ```

   Example output

   ```
   NAME             READY  UP-TO-DATE  AVAILABLE  AGE
   network-operator  1/1    1           1          56m
   ```

2. Run the following command to view the state of the Cluster Network Operator:

   ```
   $ oc get clusteroperator/network
   ```

   Example output

   ```
   NAME     VERSION  AVAILABLE  PROGRESSING  DEGRADED  SINCE
   network  4.5.4    True       False        False     50m
   ```

   The following fields provide information about the status of the operator: **AVAILABLE**, **PROGRESSING**, and **DEGRADED**. The **AVAILABLE** field is **True** when the Cluster Network Operator reports an available status condition.

5.2. VIEWING THE CLUSTER NETWORK CONFIGURATION

Every new OpenShift Container Platform installation has a **network.config** object named **cluster**.

Procedure

- Use the **oc describe** command to view the cluster network configuration:

  ```
  $ oc describe network.config/cluster
  ```

  Example output

```
Name:        cluster
Namespace:
Labels:        <none>
Annotations: <none>
API Version: config.openshift.io/v1
Kind:        Network
Metadata:
  Self Link:           /apis/config.openshift.io/v1/networks/cluster
Spec: 1
  Cluster Network:
    Cidr:        10.128.0.0/14
    Host Prefix:  23
  Network Type:   OpenShiftSDN
  Service Network:
    172.30.0.0/16
Status: 2
  Cluster Network:
    Cidr:            10.128.0.0/14
    Host Prefix:       23
  Cluster Network MTU:  8951
  Network Type:         OpenShiftSDN
  Service Network:
    172.30.0.0/16
Events: <none>
```

1   The **Spec** field displays the configured state of the cluster network.

2   The **Status** field displays the current state of the cluster network configuration.

5.3. VIEWING CLUSTER NETWORK OPERATOR STATUS

You can inspect the status and view the details of the Cluster Network Operator using the **oc describe** command.

Procedure

- Run the following command to view the status of the Cluster Network Operator:

  ```
  $ oc describe clusteroperators/network
  ```

5.4. VIEWING CLUSTER NETWORK OPERATOR LOGS

You can view Cluster Network Operator logs by using the **oc logs** command.

Procedure

- Run the following command to view the logs of the Cluster Network Operator:

  ```
  $ oc logs --namespace=openshift-network-operator deployment/network-operator
  ```

5.5. CLUSTER NETWORK OPERATOR CONFIGURATION

The configuration for the cluster network is specified as part of the Cluster Network Operator (CNO) configuration and stored in a custom resource (CR) object that is named **cluster**. The CR specifies the fields for the **Network** API in the **operator.openshift.io** API group.

The CNO configuration inherits the following fields during cluster installation from the **Network** API in the **Network.config.openshift.io** API group and these fields cannot be changed:

clusterNetwork

IP address pools from which pod IP addresses are allocated.

serviceNetwork

IP address pool for services.

defaultNetwork.type

Cluster network plugin, such as OpenShift SDN or OVN-Kubernetes.

> **NOTE**
>
> After cluster installation, you cannot modify the fields listed in the previous section.

You can specify the cluster network plugin configuration for your cluster by setting the fields for the **defaultNetwork** object in the CNO object named **cluster**.

5.5.1. Cluster Network Operator configuration object

The fields for the Cluster Network Operator (CNO) are described in the following table:

Table 5.1. Cluster Network Operator configuration object

Field	Type	Description
metadata.name	**string**	The name of the CNO object. This name is always **cluster**.
| **spec.clusterNetwork** | **array** | A list specifying the blocks of IP addresses from which pod IP addresses are allocated and the subnet prefix length assigned to each individual node in the cluster. For example:<br><br>```
spec:
  clusterNetwork:
  - cidr: 10.128.0.0/19
    hostPrefix: 23
  - cidr: 10.128.32.0/19
    hostPrefix: 23
```<br><br>This value is ready-only and inherited from the **Network.config.openshift.io** object named **cluster** during cluster installation. |

Field	Type	Description
spec.serviceNetwork	**array**	A block of IP addresses for services. The OpenShift SDN and OVN-Kubernetes network plugins support only a single IP address block for the service network. For example: ```\nspec:\n serviceNetwork:\n - 172.30.0.0/14\n``` This value is ready-only and inherited from the **Network.config.openshift.io** object named **cluster** during cluster installation.
spec.defaultNetwork	**object**	Configures the network plugin for the cluster network.
spec.kubeProxyConfig	**object**	The fields for this object specify the kube-proxy configuration. If you are using the OVN-Kubernetes cluster network plugin, the kube-proxy configuration has no effect.

defaultNetwork object configuration

The values for the **defaultNetwork** object are defined in the following table:

Table 5.2. **defaultNetwork** object

Field	Type	Description
type	**string**	Either **OpenShiftSDN** or **OVNKubernetes**. The Red Hat OpenShift Networking network plugin is selected during installation. This value cannot be changed after cluster installation. **NOTE** OpenShift Container Platform uses the OVN-Kubernetes network plugin by default.
openshiftSDNConfig	**object**	This object is only valid for the OpenShift SDN network plugin.
ovnKubernetesConfig	**object**	This object is only valid for the OVN-Kubernetes network plugin.

Configuration for the OpenShift SDN network plugin

The following table describes the configuration fields for the OpenShift SDN network plugin:

Table 5.3. **openshiftSDNConfig** object

Field	Type	Description
mode	**string**	The network isolation mode for OpenShift SDN.
mtu	**integer**	The maximum transmission unit (MTU) for the VXLAN overlay network. This value is normally configured automatically.
vxlanPort	**integer**	The port to use for all VXLAN packets. The default value is **4789**.

NOTE

You can only change the configuration for your cluster network plugin during cluster installation.

Example OpenShift SDN configuration

```
defaultNetwork:
  type: OpenShiftSDN
  openshiftSDNConfig:
    mode: NetworkPolicy
    mtu: 1450
    vxlanPort: 4789
```

Configuration for the OVN-Kubernetes network plugin
The following table describes the configuration fields for the OVN-Kubernetes network plugin:

Table 5.4. **ovnKubernetesConfig** object

Field	Type	Description
mtu	**integer**	The maximum transmission unit (MTU) for the Geneve (Generic Network Virtualization Encapsulation) overlay network. This value is normally configured automatically.
genevePort	**integer**	The UDP port for the Geneve overlay network.
ipsecConfig	**object**	If the field is present, IPsec is enabled for the cluster.
policyAuditConfig	**object**	Specify a configuration object for customizing network policy audit logging. If unset, the defaults audit log settings are used.

Field	Type	Description
gatewayConfig	**object**	Optional: Specify a configuration object for customizing how egress traffic is sent to the node gateway. **NOTE** While migrating egress traffic, you can expect some disruption to workloads and service traffic until the Cluster Network Operator (CNO) successfully rolls out the changes.

Field	Type	Description
v4InternalSubnet	If your existing network infrastructure overlaps with the **100.64.0.0/16** IPv4 subnet, you can specify a different IP address range for internal use by OVN-Kubernetes. You must ensure that the IP address range does not overlap with any other subnet used by your OpenShift Container Platform installation. The IP address range must be larger than the maximum number of nodes that can be added to the cluster. For example, if the **clusterNetwork.cidr** value is **10.128.0.0/14** and the **clusterNetwork.hostPrefix** value is /**23**, then the maximum number of nodes is **2^(23-14)=512**. This field cannot be changed after installation.	The default value is **100.64.0.0/16**.

Field	Type	Description
v6InternalSubnet	If your existing network infrastructure overlaps with the **fd98::/48** IPv6 subnet, you can specify a different IP address range for internal use by OVN-Kubernetes. You must ensure that the IP address range does not overlap with any other subnet used by your OpenShift Container Platform installation. The IP address range must be larger than the maximum number of nodes that can be added to the cluster. This field cannot be changed after installation.	The default value is **fd98::/48**.

Table 5.5. **policyAuditConfig** object

Field	Type	Description
rateLimit	integer	The maximum number of messages to generate every second per node. The default value is **20** messages per second.
maxFileSize	integer	The maximum size for the audit log in bytes. The default value is **50000000** or 50 MB.
maxLogFiles	integer	The maximum number of log files that are retained.

Field	Type	Description
destination	string	One of the following additional audit log targets: **libc** The libc **syslog()** function of the journald process on the host. **udp:\<host\>:\<port\>** A syslog server. Replace **\<host\>:\<port\>** with the host and port of the syslog server. **unix:\<file\>** A Unix Domain Socket file specified by **\<file\>**. **null** Do not send the audit logs to any additional target.
syslogFacility	string	The syslog facility, such as **kern**, as defined by RFC5424. The default value is **local0**.

Table 5.6. **gatewayConfig** object

Field	Type	Description
routingViaHost	**boolean**	Set this field to **true** to send egress traffic from pods to the host networking stack. For highly-specialized installations and applications that rely on manually configured routes in the kernel routing table, you might want to route egress traffic to the host networking stack. By default, egress traffic is processed in OVN to exit the cluster and is not affected by specialized routes in the kernel routing table. The default value is **false**. This field has an interaction with the Open vSwitch hardware offloading feature. If you set this field to **true**, you do not receive the performance benefits of the offloading because egress traffic is processed by the host networking stack.

> **NOTE**
>
> You can only change the configuration for your cluster network plugin during cluster installation, except for the **gatewayConfig** field that can be changed at runtime as a postinstallation activity.

Example OVN-Kubernetes configuration with IPSec enabled

```
defaultNetwork:
  type: OVNKubernetes
  ovnKubernetesConfig:
    mtu: 1400
    genevePort: 6081
    ipsecConfig: {}
```

kubeProxyConfig object configuration

The values for the **kubeProxyConfig** object are defined in the following table:

Table 5.7. kubeProxyConfig object

Field	Type	Description
iptablesSyncPeriod	**string**	The refresh period for **iptables** rules. The default value is **30s**. Valid suffixes include **s**, **m**, and **h** and are described in the Go **time** package documentation. **NOTE** Because of performance improvements introduced in OpenShift Container Platform 4.3 and greater, adjusting the **iptablesSyncPeriod** parameter is no longer necessary.
proxyArguments.iptables-min-sync-period	**array**	The minimum duration before refreshing **iptables** rules. This field ensures that the refresh does not happen too frequently. Valid suffixes include **s**, **m**, and **h** and are described in the Go **time** package. The default value is: ```\nkubeProxyConfig:\n proxyArguments:\n iptables-min-sync-period:\n - 0s\n```

5.5.2. Cluster Network Operator example configuration

A complete CNO configuration is specified in the following example:

Example Cluster Network Operator object

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  clusterNetwork: ①
  - cidr: 10.128.0.0/14
    hostPrefix: 23
  serviceNetwork: ②
  - 172.30.0.0/16
  defaultNetwork: ③
    type: OpenShiftSDN
    openshiftSDNConfig:
      mode: NetworkPolicy
      mtu: 1450
```

```
      vxlanPort: 4789
kubeProxyConfig:
  iptablesSyncPeriod: 30s
  proxyArguments:
    iptables-min-sync-period:
    - 0s
```

1 2 3 Configured only during cluster installation.

5.6. ADDITIONAL RESOURCES

- **Network** API in the **operator.openshift.io** API group

CHAPTER 6. DNS OPERATOR IN OPENSHIFT CONTAINER PLATFORM

The DNS Operator deploys and manages CoreDNS to provide a name resolution service to pods, enabling DNS–based Kubernetes Service discovery in OpenShift Container Platform.

6.1. DNS OPERATOR

The DNS Operator implements the **dns** API from the **operator.openshift.io** API group. The Operator deploys CoreDNS using a daemon set, creates a service for the daemon set, and configures the kubelet to instruct pods to use the CoreDNS service IP address for name resolution.

Procedure

The DNS Operator is deployed during installation with a **Deployment** object.

1. Use the **oc get** command to view the deployment status:

   ```
   $ oc get -n openshift-dns-operator deployment/dns-operator
   ```

   Example output

   ```
   NAME         READY    UP-TO-DATE  AVAILABLE  AGE
   dns-operator 1/1      1           1          23h
   ```

2. Use the **oc get** command to view the state of the DNS Operator:

   ```
   $ oc get clusteroperator/dns
   ```

   Example output

   ```
   NAME    VERSION    AVAILABLE  PROGRESSING  DEGRADED  SINCE
   dns     4.1.0-0.11 True       False        False     92m
   ```

   AVAILABLE, **PROGRESSING** and **DEGRADED** provide information about the status of the operator. **AVAILABLE** is **True** when at least 1 pod from the CoreDNS daemon set reports an **Available** status condition.

6.2. CHANGING THE DNS OPERATOR MANAGEMENTSTATE

DNS manages the CoreDNS component to provide a name resolution service for pods and services in the cluster. The **managementState** of the DNS Operator is set to **Managed** by default, which means that the DNS Operator is actively managing its resources. You can change it to **Unmanaged**, which means the DNS Operator is not managing its resources.

The following are use cases for changing the DNS Operator **managementState**:

- You are a developer and want to test a configuration change to see if it fixes an issue in CoreDNS. You can stop the DNS Operator from overwriting the fix by setting the **managementState** to **Unmanaged**.

- You are a cluster administrator and have reported an issue with CoreDNS, but need to apply a workaround until the issue is fixed. You can set the **managementState** field of the DNS Operator to **Unmanaged** to apply the workaround.

Procedure

- Change **managementState** DNS Operator:

  ```
  oc patch dns.operator.openshift.io default --type merge --patch '{"spec":
  {"managementState":"Unmanaged"}}'
  ```

6.3. CONTROLLING DNS POD PLACEMENT

The DNS Operator has two daemon sets: one for CoreDNS and one for managing the **/etc/hosts** file. The daemon set for **/etc/hosts** must run on every node host to add an entry for the cluster image registry to support pulling images. Security policies can prohibit communication between pairs of nodes, which prevents the daemon set for CoreDNS from running on every node.

As a cluster administrator, you can use a custom node selector to configure the daemon set for CoreDNS to run or not run on certain nodes.

Prerequisites

- You installed the **oc** CLI.

- You are logged in to the cluster with a user with **cluster-admin** privileges.

Procedure

- To prevent communication between certain nodes, configure the **spec.nodePlacement.nodeSelector** API field:

  1. Modify the DNS Operator object named **default**:

     ```
     $ oc edit dns.operator/default
     ```

  2. Specify a node selector that includes only control plane nodes in the **spec.nodePlacement.nodeSelector** API field:

     ```
     spec:
       nodePlacement:
         nodeSelector:
           node-role.kubernetes.io/worker: ""
     ```

- To allow the daemon set for CoreDNS to run on nodes, configure a taint and toleration:

  1. Modify the DNS Operator object named **default**:

     ```
     $ oc edit dns.operator/default
     ```

  2. Specify a taint key and a toleration for the taint:

     ```
     spec:
       nodePlacement:
     ```

```
      tolerations:
      - effect: NoExecute
        key: "dns-only"
        operators: Equal
        value: abc
        tolerationSeconds: 3600 ❶
```

❶ If the taint is **dns-only**, it can be tolerated indefinitely. You can omit **tolerationSeconds**.

6.4. VIEW THE DEFAULT DNS

Every new OpenShift Container Platform installation has a **dns.operator** named **default**.

Procedure

1. Use the **oc describe** command to view the default **dns**:

   ```
   $ oc describe dns.operator/default
   ```

   Example output

   ```
   Name:         default
   Namespace:
   Labels:       <none>
   Annotations:  <none>
   API Version:  operator.openshift.io/v1
   Kind:         DNS
   ...
   Status:
     Cluster Domain:  cluster.local ❶
     Cluster IP:      172.30.0.10 ❷
   ...
   ```

   ❶ The Cluster Domain field is the base DNS domain used to construct fully qualified pod and service domain names.

   ❷ The Cluster IP is the address pods query for name resolution. The IP is defined as the 10th address in the service CIDR range.

2. To find the service CIDR of your cluster, use the **oc get** command:

   ```
   $ oc get networks.config/cluster -o jsonpath='{$.status.serviceNetwork}'
   ```

Example output

```
[172.30.0.0/16]
```

6.5. USING DNS FORWARDING

You can use DNS forwarding to override the default forwarding configuration in the **/etc/resolv.conf** file in the following ways:

- Specify name servers for every zone. If the forwarded zone is the Ingress domain managed by OpenShift Container Platform, then the upstream name server must be authorized for the domain.

- Provide a list of upstream DNS servers.

- Change the default forwarding policy.

> **NOTE**
>
> A DNS forwarding configuration for the default domain can have both the default servers specified in the **/etc/resolv.conf** file and the upstream DNS servers.

Procedure

1. Modify the DNS Operator object named **default**:

   ```
   $ oc edit dns.operator/default
   ```

   After you issue the previous command, the Operator creates and updates the config map named **dns-default** with additional server configuration blocks based on **Server**. If none of the servers have a zone that matches the query, then name resolution falls back to the upstream DNS servers.

   Configuring DNS forwarding

   ```
   apiVersion: operator.openshift.io/v1
   kind: DNS
   metadata:
     name: default
   spec:
     servers:
     - name: example-server 1
       zones: 2
       - example.com
       forwardPlugin:
         policy: Random 3
         upstreams: 4
         - 1.1.1.1
         - 2.2.2.2:5353
     upstreamResolvers: 5
       policy: Random 6
       upstreams: 7
       - type: SystemResolvConf 8
       - type: Network
         address: 1.2.3.4 9
         port: 53 10
   ```

   1   Must comply with the **rfc6335** service name syntax.

2 Must conform to the definition of a subdomain in the **rfc1123** service name syntax. The cluster domain, **cluster.local**, is an invalid subdomain for the **zones** field.

3 Defines the policy to select upstream resolvers. Default value is **Random**. You can also use the values **RoundRobin**, and **Sequential**.

4 A maximum of 15 **upstreams** is allowed per **forwardPlugin**.

5 Optional. You can use it to override the default policy and forward DNS resolution to the specified DNS resolvers (upstream resolvers) for the default domain. If you do not provide any upstream resolvers, the DNS name queries go to the servers in **/etc/resolv.conf**.

6 Determines the order in which upstream servers are selected for querying. You can specify one of these values: **Random**, **RoundRobin**, or **Sequential**. The default value is **Sequential**.

7 Optional. You can use it to provide upstream resolvers.

8 You can specify two types of **upstreams** – **SystemResolvConf** and **Network**. **SystemResolvConf** configures the upstream to use **/etc/resolv.conf** and **Network** defines a **Networkresolver**. You can specify one or both.

9 If the specified type is **Network**, you must provide an IP address. The **address** field must be a valid IPv4 or IPv6 address.

10 If the specified type is **Network**, you can optionally provide a port. The **port** field must have a value between **1** and **65535**. If you do not specify a port for the upstream, by default port 853 is tried.

2. Optional: When working in a highly regulated environment, you might need the ability to secure DNS traffic when forwarding requests to upstream resolvers so that you can ensure additional DNS traffic and data privacy. Cluster administrators can configure transport layer security (TLS) for forwarded DNS queries.

   Configuring DNS forwarding with TLS

```
apiVersion: operator.openshift.io/v1
kind: DNS
metadata:
  name: default
spec:
  servers:
  - name: example-server 1
    zones: 2
    - example.com
    forwardPlugin:
      transportConfig:
        transport: TLS 3
        tls:
          caBundle:
            name: mycacert
          serverName: dnstls.example.com 4
      policy: Random 5
      upstreams: 6
      - 1.1.1.1
```

```
      - 2.2.2.2:5353
upstreamResolvers: 7
 transportConfig:
   transport: TLS
   tls:
     caBundle:
       name: mycacert
     serverName: dnstls.example.com
 upstreams:
 - type: Network 8
     address: 1.2.3.4 9
     port: 53 10
```

1 Must comply with the **rfc6335** service name syntax.

2 Must conform to the definition of a subdomain in the **rfc1123** service name syntax. The cluster domain, **cluster.local**, is an invalid subdomain for the **zones** field. The cluster domain, **cluster.local**, is an invalid **subdomain** for **zones**.

3 When configuring TLS for forwarded DNS queries, set the **transport** field to have the value **TLS**. By default, CoreDNS caches forwarded connections for 10 seconds. CoreDNS will hold a TCP connection open for those 10 seconds if no request is issued. With large clusters, ensure that your DNS server is aware that it might get many new connections to hold open because you can initiate a connection per node. Set up your DNS hierarchy accordingly to avoid performance issues.

4 When configuring TLS for forwarded DNS queries, this is a mandatory server name used as part of the server name indication (SNI) to validate the upstream TLS server certificate.

5 Defines the policy to select upstream resolvers. Default value is **Random**. You can also use the values **RoundRobin**, and **Sequential**.

6 Required. You can use it to provide upstream resolvers. A maximum of 15 **upstreams** entries are allowed per **forwardPlugin** entry.

7 Optional. You can use it to override the default policy and forward DNS resolution to the specified DNS resolvers (upstream resolvers) for the default domain. If you do not provide any upstream resolvers, the DNS name queries go to the servers in **/etc/resolv.conf**.

8 **Network** type indicates that this upstream resolver should handle forwarded requests separately from the upstream resolvers listed in **/etc/resolv.conf**. Only the **Network** type is allowed when using TLS and you must provide an IP address.

9 The **address** field must be a valid IPv4 or IPv6 address.

10 You can optionally provide a port. The **port** must have a value between **1** and **65535**. If you do not specify a port for the upstream, by default port 853 is tried.

> **NOTE**
>
> If **servers** is undefined or invalid, the config map only contains the default server.

Verification

1. View the config map:

```
$ oc get configmap/dns-default -n openshift-dns -o yaml
```

Sample DNS ConfigMap based on previous sample DNS

```
apiVersion: v1
data:
  Corefile: |
    example.com:5353 {
        forward . 1.1.1.1 2.2.2.2:5353
    }
    bar.com:5353 example.com:5353 {
        forward . 3.3.3.3 4.4.4.4:5454 ❶
    }
    .:5353 {
        errors
        health
        kubernetes cluster.local in-addr.arpa ip6.arpa {
           pods insecure
           upstream
           fallthrough in-addr.arpa ip6.arpa
        }
        prometheus :9153
        forward . /etc/resolv.conf 1.2.3.4:53 {
           policy Random
        }
        cache 30
        reload
    }
kind: ConfigMap
metadata:
  labels:
    dns.operator.openshift.io/owning-dns: default
  name: dns-default
  namespace: openshift-dns
```

❶ Changes to the **forwardPlugin** triggers a rolling update of the CoreDNS daemon set.

Additional resources

- For more information on DNS forwarding, see the CoreDNS forward documentation.

6.6. DNS OPERATOR STATUS

You can inspect the status and view the details of the DNS Operator using the **oc describe** command.

Procedure

View the status of the DNS Operator:

```
$ oc describe clusteroperators/dns
```

6.7. DNS OPERATOR LOGS

You can view DNS Operator logs by using the **oc logs** command.

Procedure

View the logs of the DNS Operator:

```
$ oc logs -n openshift-dns-operator deployment/dns-operator -c dns-operator
```

6.8. SETTING THE COREDNS LOG LEVEL

You can configure the CoreDNS log level to determine the amount of detail in logged error messages. The valid values for CoreDNS log level are **Normal**, **Debug**, and **Trace**. The default **logLevel** is **Normal**.

> **NOTE**
>
> The errors plugin is always enabled. The following **logLevel** settings report different error responses:
>
> - **logLevel**: **Normal** enables the "errors" class: **log . { class error }**.
>
> - **logLevel**: **Debug** enables the "denial" class: **log . { class denial error }**.
>
> - **logLevel**: **Trace** enables the "all" class: **log . { class all }**.

Procedure

- To set **logLevel** to **Debug**, enter the following command:

  ```
  $ oc patch dnses.operator.openshift.io/default -p '{"spec":{"logLevel":"Debug"}}' --type=merge
  ```

- To set **logLevel** to **Trace**, enter the following command:

  ```
  $ oc patch dnses.operator.openshift.io/default -p '{"spec":{"logLevel":"Trace"}}' --type=merge
  ```

Verification

- To ensure the desired log level was set, check the config map:

  ```
  $ oc get configmap/dns-default -n openshift-dns -o yaml
  ```

6.9. SETTING THE COREDNS OPERATOR LOG LEVEL

Cluster administrators can configure the Operator log level to more quickly track down OpenShift DNS issues. The valid values for **operatorLogLevel** are **Normal**, **Debug**, and **Trace**. **Trace** has the most detailed information. The default **operatorlogLevel** is **Normal**. There are seven logging levels for issues: Trace, Debug, Info, Warning, Error, Fatal and Panic. After the logging level is set, log entries with that severity or anything above it will be logged.

- **operatorLogLevel: "Normal"** sets **logrus.SetLogLevel("Info")**.

- **operatorLogLevel: "Debug"** sets **logrus.SetLogLevel("Debug")**.

- **operatorLogLevel: "Trace"** sets **logrus.SetLogLevel("Trace")**.

Procedure

- To set **operatorLogLevel** to **Debug**, enter the following command:

```
$ oc patch dnses.operator.openshift.io/default -p '{"spec":{"operatorLogLevel":"Debug"}}' --
type=merge
```

- To set **operatorLogLevel** to **Trace**, enter the following command:

```
$ oc patch dnses.operator.openshift.io/default -p '{"spec":{"operatorLogLevel":"Trace"}}' --
type=merge
```

6.10. TUNING THE COREDNS CACHE

You can configure the maximum duration of both successful or unsuccessful caching, also known as positive or negative caching respectively, done by CoreDNS. Tuning the duration of caching of DNS query responses can reduce the load for any upstream DNS resolvers.

Procedure

1. Edit the DNS Operator object named **default** by running the following command:

```
$ oc edit dns.operator.openshift.io/default
```

2. Modify the time-to-live (TTL) caching values:

Configuring DNS caching

```
apiVersion: operator.openshift.io/v1
kind: DNS
metadata:
  name: default
spec:
  cache:
    positiveTTL: 1h    1
    negativeTTL: 0.5h10m    2
```

1 The string value **1h** is converted to its respective number of seconds by CoreDNS. If this field is omitted, the value is assumed to be **0s** and the cluster uses the internal default value of **900s** as a fallback.

2 The string value can be a combination of units such as **0.5h10m** and is converted to its respective number of seconds by CoreDNS. If this field is omitted, the value is assumed to be **0s** and the cluster uses the internal default value of **30s** as a fallback.

> **WARNING**
>
> Setting TTL fields to low values could lead to an increased load on the cluster, any upstream resolvers, or both.

CHAPTER 7. INGRESS OPERATOR IN OPENSHIFT CONTAINER PLATFORM

7.1. OPENSHIFT CONTAINER PLATFORM INGRESS OPERATOR

When you create your OpenShift Container Platform cluster, pods and services running on the cluster are each allocated their own IP addresses. The IP addresses are accessible to other pods and services running nearby but are not accessible to outside clients. The Ingress Operator implements the **IngressController** API and is the component responsible for enabling external access to OpenShift Container Platform cluster services.

The Ingress Operator makes it possible for external clients to access your service by deploying and managing one or more HAProxy-based Ingress Controllers to handle routing. You can use the Ingress Operator to route traffic by specifying OpenShift Container Platform **Route** and Kubernetes **Ingress** resources. Configurations within the Ingress Controller, such as the ability to define **endpointPublishingStrategy** type and internal load balancing, provide ways to publish Ingress Controller endpoints.

7.2. THE INGRESS CONFIGURATION ASSET

The installation program generates an asset with an **Ingress** resource in the **config.openshift.io** API group, **cluster-ingress-02-config.yml**.

YAML Definition of the **Ingress** resource

```
apiVersion: config.openshift.io/v1
kind: Ingress
metadata:
  name: cluster
spec:
  domain: apps.openshiftdemos.com
```

The installation program stores this asset in the **cluster-ingress-02-config.yml** file in the **manifests/** directory. This **Ingress** resource defines the cluster-wide configuration for Ingress. This Ingress configuration is used as follows:

- The Ingress Operator uses the domain from the cluster Ingress configuration as the domain for the default Ingress Controller.

- The OpenShift API Server Operator uses the domain from the cluster Ingress configuration. This domain is also used when generating a default host for a **Route** resource that does not specify an explicit host.

7.3. INGRESS CONTROLLER CONFIGURATION PARAMETERS

The **ingresscontrollers.operator.openshift.io** resource offers the following configuration parameters.

Parameter	Description

Parameter	Description
domain	**domain** is a DNS name serviced by the Ingress Controller and is used to configure multiple features: • For the **LoadBalancerService** endpoint publishing strategy, **domain** is used to configure DNS records. See **endpointPublishingStrategy**. • When using a generated default certificate, the certificate is valid for **domain** and its **subdomains**. See **defaultCertificate**. • The value is published to individual Route statuses so that users know where to target external DNS records. The **domain** value must be unique among all Ingress Controllers and cannot be updated. If empty, the default value is **ingress.config.openshift.io/cluster .spec.domain**.
replicas	**replicas** is the desired number of Ingress Controller replicas. If not set, the default value is **2**.
endpointPublishingStrategy	**endpointPublishingStrategy** is used to publish the Ingress Controller endpoints to other networks, enable load balancer integrations, and provide access to other systems. On GCP, AWS, and Azure you can configure the following **endpointPublishingStrategy** fields: • **loadBalancer.scope** • **loadBalancer.allowedSourceRanges** If not set, the default value is based on **infrastructure.config.openshift.io/cluster .status.platform**: • Amazon Web Services (AWS): **LoadBalancerService** (with External scope) • Azure: **LoadBalancerService** (with External scope) • Google Cloud Platform (GCP): **LoadBalancerService** (with External scope) • Bare metal: **NodePortService** • Other: **HostNetwork**

Parameter	Description

NOTE

HostNetwork has a **hostNetwork** field with the following default values for the optional binding ports: **httpPort: 80**, **httpsPort: 443**, and **statsPort: 1936**. With the binding ports, you can deploy multiple Ingress Controllers on the same node for the **HostNetwork** strategy.

Example

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: internal
  namespace: openshift-ingress-operator
spec:
  domain: example.com
  endpointPublishingStrategy:
    type: HostNetwork
    hostNetwork:
      httpPort: 80
      httpsPort: 443
      statsPort: 1936
```

NOTE

On Red Hat OpenStack Platform (RHOSP), the **LoadBalancerService** endpoint publishing strategy is only supported if a cloud provider is configured to create health monitors. For RHOSP 16.2, this strategy is only possible if you use the Amphora Octavia provider.

For more information, see the "Setting cloud provider options" section of the RHOSP installation documentation.

For most platforms, the **endpointPublishingStrategy** value can be updated. On GCP, you can configure the following **endpointPublishingStrategy** fields:

- **loadBalancer.scope**

- **loadbalancer.providerParameters.gcp.clientAccess**

- **hostNetwork.protocol**

- **nodePort.protocol**

Parameter	Description
defaultCertificate	The **defaultCertificate** value is a reference to a secret that contains the default certificate that is served by the Ingress Controller. When Routes do not specify their own certificate, **defaultCertificate** is used. The secret must contain the following keys and data: * **tls.crt**: certificate file contents * **tls.key**: key file contents If not set, a wildcard certificate is automatically generated and used. The certificate is valid for the Ingress Controller **domain** and **subdomains**, and the generated certificate's CA is automatically integrated with the cluster's trust store. The in-use certificate, whether generated or user-specified, is automatically integrated with OpenShift Container Platform built-in OAuth server.
namespaceSelector	**namespaceSelector** is used to filter the set of namespaces serviced by the Ingress Controller. This is useful for implementing shards.
routeSelector	**routeSelector** is used to filter the set of Routes serviced by the Ingress Controller. This is useful for implementing shards.

nodePlacement	**nodePlacement** enables explicit control over the scheduling of the Ingress Controller.
	
	If not set, the defaults values are used.
	
	NOTE
	
	The **nodePlacement** parameter includes two parts, **nodeSelector** and **tolerations**. For example:
	
	<pre>nodePlacement:
	 nodeSelector:
	   matchLabels:
	     kubernetes.io/os: linux
	 tolerations:
	 - effect: NoSchedule
	   operator: Exists</pre>

Parameter	Description
tlsSecurityProfile	**tlsSecurityProfile** specifies settings for TLS connections for Ingress Controllers.
	If not set, the default value is based on the **apiservers.config.openshift.io/cluster** resource.
	When using the **Old**, **Intermediate**, and **Modern** profile types, the effective profile configuration is subject to change between releases. For example, given a specification to use the **Intermediate** profile deployed on release **X.Y.Z**, an upgrade to release **X.Y.Z+1** may cause a new profile configuration to be applied to the Ingress Controller, resulting in a rollout.
	The minimum TLS version for Ingress Controllers is **1.1**, and the maximum TLS version is **1.3**.
	NOTE Ciphers and the minimum TLS version of the configured security profile are reflected in the **TLSProfile** status.
	IMPORTANT The Ingress Operator converts the TLS **1.0** of an **Old** or **Custom** profile to **1.1**.
clientTLS	**clientTLS** authenticates client access to the cluster and services; as a result, mutual TLS authentication is enabled. If not set, then client TLS is not enabled.
	clientTLS has the required subfields, **spec.clientTLS.clientCertificatePolicy** and **spec.clientTLS.ClientCA**.
	The **ClientCertificatePolicy** subfield accepts one of the two values: **Required** or **Optional**. The **ClientCA** subfield specifies a config map that is in the openshift-config namespace. The config map should contain a CA certificate bundle.
	The **AllowedSubjectPatterns** is an optional value that specifies a list of regular expressions, which are matched against the distinguished name on a valid client certificate to filter requests. The regular expressions must use PCRE syntax. At least one pattern must match a client certificate's distinguished name; otherwise, the Ingress Controller rejects the certificate and denies the connection. If not specified, the Ingress Controller does not reject certificates based on the distinguished name.

Parameter	Description
routeAdmission	**routeAdmission** defines a policy for handling new route claims, such as allowing or denying claims across namespaces.
	namespaceOwnership describes how hostname claims across namespaces should be handled. The default is **Strict**.
	Strict: does not allow routes to claim the same hostname across namespaces. **InterNamespaceAllowed**: allows routes to claim different paths of the same hostname across namespaces.
	wildcardPolicy describes how routes with wildcard policies are handled by the Ingress Controller.
	WildcardsAllowed: Indicates routes with any wildcard policy are admitted by the Ingress Controller. **WildcardsDisallowed**: Indicates only routes with a wildcard policy of **None** are admitted by the Ingress Controller. Updating **wildcardPolicy** from **WildcardsAllowed** to **WildcardsDisallowed** causes admitted routes with a wildcard policy of **Subdomain** to stop working. These routes must be recreated to a wildcard policy of **None** to be readmitted by the Ingress Controller. **WildcardsDisallowed** is the default setting.

Parameter	Description

IngressControllerLogging	**logging** defines parameters for what is logged where. If this field is empty, operational logs are enabled but access logs are disabled.
	
	• **access** describes how client requests are logged. If this field is empty, access logging is disabled.
	
	    ○ **destination** describes a destination for log messages.
	
	        ■ **type** is the type of destination for logs:
	
	           • **Container** specifies that logs should go to a sidecar container. The Ingress Operator configures the container, named **logs**, on the Ingress Controller pod and configures the Ingress Controller to write logs to the container. The expectation is that the administrator configures a custom logging solution that reads logs from this container. Using container logs means that logs may be dropped if the rate of logs exceeds the container runtime capacity or the custom logging solution capacity.
	
	           • **Syslog** specifies that logs are sent to a Syslog endpoint. The administrator must specify an endpoint that can receive Syslog messages. The expectation is that the administrator has configured a custom Syslog instance.
	
	        ■ **container** describes parameters for the **Container** logging destination type. Currently there are no parameters for container logging, so this field must be empty.
	
	        ■ **syslog** describes parameters for the **Syslog** logging destination type:
	
	           • **address** is the IP address of the syslog endpoint that receives log messages.
	
	           • **port** is the UDP port number of the syslog endpoint that receives log messages.
	
	           • **maxLength** is the maximum length of the syslog message. It must be between **480** and **4096** bytes. If this field is empty, the maximum length is set to the default value of **1024** bytes.
	
	           • **facility** specifies the syslog facility of log messages. If this field is empty, the facility is **local1**. Otherwise, it must specify a valid syslog facility: **kern**, **user**, **mail**, **daemon**, **auth**, **syslog**, **lpr**, **news**, **uucp**, **cron**, **auth2**, **ftp**, **ntp**, **audit**, **alert**, **cron2**, **local0**, **local1**, **local2**, **local3**. **local4**, **local5**, **local6**, or **local7**.
	
	    ○ **httpLogFormat** specifies the format of the log message for an HTTP request. If this field is empty, log messages use the implementation's default HTTP log format. For HAProxy's default HTTP log format, see the HAProxy documentation.

Parameter	Description
httpHeaders	**httpHeaders** defines the policy for HTTP headers. By setting the **forwardedHeaderPolicy** for the **IngressControllerHTTPHeaders**, you specify when and how the Ingress Controller sets the **Forwarded**, **X-Forwarded-For**, **X-Forwarded-Host**, **X-Forwarded-Port**, **X-Forwarded-Proto**, and **X-Forwarded-Proto-Version** HTTP headers. By default, the policy is set to **Append**. • **Append** specifies that the Ingress Controller appends the headers, preserving any existing headers. • **Replace** specifies that the Ingress Controller sets the headers, removing any existing headers. • **IfNone** specifies that the Ingress Controller sets the headers if they are not already set. • **Never** specifies that the Ingress Controller never sets the headers, preserving any existing headers. By setting **headerNameCaseAdjustments**, you can specify case adjustments that can be applied to HTTP header names. Each adjustment is specified as an HTTP header name with the desired capitalization. For example, specifying **X-Forwarded-For** indicates that the **x-forwarded-for** HTTP header should be adjusted to have the specified capitalization. These adjustments are only applied to cleartext, edge-terminated, and re-encrypt routes, and only when using HTTP/1. For request headers, these adjustments are applied only for routes that have the **haproxy.router.openshift.io/h1-adjust-case=true** annotation. For response headers, these adjustments are applied to all HTTP responses. If this field is empty, no request headers are adjusted.
httpCompression	**httpCompression** defines the policy for HTTP traffic compression. • **mimeTypes** defines a list of MIME types to which compression should be applied. For example, **text/css; charset=utf-8**, **text/html**, **text/***, **image/svg+xml**, **application/octet-stream**, **X-custom/customsub**, using the format pattern, **type/subtype; [;attribute=value]**. The **types** are: application, image, message, multipart, text, video, or a custom type prefaced by **X-**; e.g. To see the full notation for MIME types and subtypes, see RFC1341
httpErrorCodePages	**httpErrorCodePages** specifies custom HTTP error code response pages. By default, an IngressController uses error pages built into the IngressController image.

Parameter	Description

httpCaptureCookies	**httpCaptureCookies** specifies HTTP cookies that you want to capture in access logs. If the **httpCaptureCookies** field is empty, the access logs do not capture the cookies.
	
	For any cookie that you want to capture, the following parameters must be in your **IngressController** configuration:
	
	• **name** specifies the name of the cookie.
	
	• **maxLength** specifies tha maximum length of the cookie.
	
	• **matchType** specifies if the field **name** of the cookie exactly matches the capture cookie setting or is a prefix of the capture cookie setting. The **matchType** field uses the **Exact** and **Prefix** parameters.
	
	For example:
	
	<pre>httpCaptureCookies:
	- matchType: Exact
	  maxLength: 128
	  name: MYCOOKIE</pre>

httpCaptureHeaders	**httpCaptureHeaders** specifies the HTTP headers that you want to capture in the access logs. If the **httpCaptureHeaders** field is empty, the access logs do not capture the headers.
	
	httpCaptureHeaders contains two lists of headers to capture in the access logs. The two lists of header fields are **request** and **response**. In both lists, the **name** field must specify the header name and the **maxlength** field must specify the maximum length of the header. For example:
	
	<pre>httpCaptureHeaders:
	  request:
	  - maxLength: 256
	    name: Connection
	  - maxLength: 128
	    name: User-Agent
	  response:
	  - maxLength: 256
	    name: Content-Type
	  - maxLength: 256
	    name: Content-Length</pre>

tuningOptions	**tuningOptions** specifies options for tuning the performance of Ingress Controller pods. • **clientFinTimeout** specifies how long a connection is held open while waiting for the client response to the server closing the connection. The default timeout is **1s**. • **clientTimeout** specifies how long a connection is held open while waiting for a client response. The default timeout is **30s**. • **headerBufferBytes** specifies how much memory is reserved, in

Parameter	Description
	bytes, for Ingress Controller connection sessions. This value must be at least **16384** if HTTP/2 is enabled for the Ingress Controller. If not set, the default value is **32768** bytes. Setting this field not recommended because **headerBufferBytes** values that are too small can break the Ingress Controller, and **headerBufferBytes** values that are too large could cause the Ingress Controller to use significantly more memory than necessary.

- **headerBufferMaxRewriteBytes** specifies how much memory should be reserved, in bytes, from **headerBufferBytes** for HTTP header rewriting and appending for Ingress Controller connection sessions. The minimum value for **headerBufferMaxRewriteBytes** is **4096**. **headerBufferBytes** must be greater than **headerBufferMaxRewriteBytes** for incoming HTTP requests. If not set, the default value is **8192** bytes. Setting this field not recommended because **headerBufferMaxRewriteBytes** values that are too small can break the Ingress Controller and **headerBufferMaxRewriteBytes** values that are too large could cause the Ingress Controller to use significantly more memory than necessary.

- **healthCheckInterval** specifies how long the router waits between health checks. The default is **5s**.

- **serverFinTimeout** specifies how long a connection is held open while waiting for the server response to the client that is closing the connection. The default timeout is **1s**.

- **serverTimeout** specifies how long a connection is held open while waiting for a server response. The default timeout is **30s**.

- **threadCount** specifies the number of threads to create per HAProxy process. Creating more threads allows each Ingress Controller pod to handle more connections, at the cost of more system resources being used. HAProxy supports up to **64** threads. If this field is empty, the Ingress Controller uses the default value of **4** threads. The default value can change in future releases. Setting this field is not recommended because increasing the number of HAProxy threads allows Ingress Controller pods to use more CPU time under load, and prevent other pods from receiving the CPU resources they need to perform. Reducing the number of threads can cause the Ingress Controller to perform poorly.

- **tlsInspectDelay** specifies how long the router can hold data to find a matching route. Setting this value too short can cause the router to fall back to the default certificate for edge-terminated, reencrypted, or passthrough routes, even when using a better matched certificate. The default inspect delay is **5s**.

- **tunnelTimeout** specifies how long a tunnel connection, including websockets, remains open while the tunnel is idle. The default timeout is **1h**.

- **maxConnections** specifies the maximum number of simultaneous connections that can be established per HAProxy process. Increasing this value allows each ingress controller pod to handle more connections at the cost of additional system resources. Permitted values are **0**, **-1**, any value within the range **2000** and **2000000**, or the field can be left empty.

  - If this field is left empty or has the value **0**, the Ingress Controller will use the default value of **50000**. This value is subject to change in future releases.

  - If the field has the value of **-1**, then HAProxy will dynamically compute a maximum value based on the available **ulimits** in the

Parameter	Description
	running container. This process results in a large computed value that will incur significant memory usage compared to the current default value of **50000**. ◦ If the field has a value that is greater than the current operating system limit, the HAProxy process will not start. ◦ If you choose a discrete value and the router pod is migrated to a new node, it is possible the new node does not have an identical **ulimit** configured. In such cases, the pod fails to start. ◦ If you have nodes with different **ulimits** configured, and you choose a discrete value, it is recommended to use the value of **-1** for this field so that the maximum number of connections is calculated at runtime.
logEmptyRequests	**logEmptyRequests** specifies connections for which no request is received and logged. These empty requests come from load balancer health probes or web browser speculative connections (preconnect) and logging these requests can be undesirable. However, these requests can be caused by network errors, in which case logging empty requests can be useful for diagnosing the errors. These requests can be caused by port scans, and logging empty requests can aid in detecting intrusion attempts. Allowed values for this field are **Log** and **Ignore**. The default value is **Log**. The **LoggingPolicy** type accepts either one of two values: ● **Log**: Setting this value to **Log** indicates that an event should be logged. ● **Ignore**: Setting this value to **Ignore** sets the **dontlognull** option in the HAproxy configuration.
HTTPEmptyRequestsPolicy	**HTTPEmptyRequestsPolicy** describes how HTTP connections are handled if the connection times out before a request is received. Allowed values for this field are **Respond** and **Ignore**. The default value is **Respond**. The **HTTPEmptyRequestsPolicy** type accepts either one of two values: ● **Respond**: If the field is set to **Respond**, the Ingress Controller sends an HTTP **400** or **408** response, logs the connection if access logging is enabled, and counts the connection in the appropriate metrics. ● **Ignore**: Setting this option to **Ignore** adds the **http-ignore-probes** parameter in the HAproxy configuration. If the field is set to **Ignore**, the Ingress Controller closes the connection without sending a response, then logs the connection, or incrementing metrics. These connections come from load balancer health probes or web browser speculative connections (preconnect) and can be safely ignored. However, these requests can be caused by network errors, so setting this field to **Ignore** can impede detection and diagnosis of problems. These requests can be caused by port scans, in which case logging empty requests can aid in detecting intrusion attempts.

NOTE

All parameters are optional.

7.3.1. Ingress Controller TLS security profiles

TLS security profiles provide a way for servers to regulate which ciphers a connecting client can use when connecting to the server.

7.3.1.1. Understanding TLS security profiles

You can use a TLS (Transport Layer Security) security profile to define which TLS ciphers are required by various OpenShift Container Platform components. The OpenShift Container Platform TLS security profiles are based on Mozilla recommended configurations.

You can specify one of the following TLS security profiles for each component:

Table 7.1. TLS security profiles

Profile	Description
Old	This profile is intended for use with legacy clients or libraries. The profile is based on the Old backward compatibility recommended configuration. The **Old** profile requires a minimum TLS version of 1.0. **NOTE** For the Ingress Controller, the minimum TLS version is converted from 1.0 to 1.1.
Intermediate	This profile is the recommended configuration for the majority of clients. It is the default TLS security profile for the Ingress Controller, kubelet, and control plane. The profile is based on the Intermediate compatibility recommended configuration. The **Intermediate** profile requires a minimum TLS version of 1.2.
Modern	This profile is intended for use with modern clients that have no need for backwards compatibility. This profile is based on the Modern compatibility recommended configuration. The **Modern** profile requires a minimum TLS version of 1.3.
Custom	This profile allows you to define the TLS version and ciphers to use. **WARNING** Use caution when using a **Custom** profile, because invalid configurations can cause problems.

> **NOTE**
>
> When using one of the predefined profile types, the effective profile configuration is subject to change between releases. For example, given a specification to use the Intermediate profile deployed on release X.Y.Z, an upgrade to release X.Y.Z+1 might cause a new profile configuration to be applied, resulting in a rollout.

7.3.1.2. Configuring the TLS security profile for the Ingress Controller

To configure a TLS security profile for an Ingress Controller, edit the **IngressController** custom resource (CR) to specify a predefined or custom TLS security profile. If a TLS security profile is not configured, the default value is based on the TLS security profile set for the API server.

Sample IngressController CR that configures the Old TLS security profile

```
apiVersion: operator.openshift.io/v1
kind: IngressController
 ...
spec:
  tlsSecurityProfile:
    old: {}
    type: Old
 ...
```

The TLS security profile defines the minimum TLS version and the TLS ciphers for TLS connections for Ingress Controllers.

You can see the ciphers and the minimum TLS version of the configured TLS security profile in the **IngressController** custom resource (CR) under **Status.Tls Profile** and the configured TLS security profile under **Spec.Tls Security Profile**. For the **Custom** TLS security profile, the specific ciphers and minimum TLS version are listed under both parameters.

> **NOTE**
>
> The HAProxy Ingress Controller image supports TLS **1.3** and the **Modern** profile.
>
> The Ingress Operator also converts the TLS **1.0** of an **Old** or **Custom** profile to **1.1**.

Prerequisites

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

Procedure

1. Edit the **IngressController** CR in the **openshift-ingress-operator** project to configure the TLS security profile:

   ```
   $ oc edit IngressController default -n openshift-ingress-operator
   ```

2. Add the **spec.tlsSecurityProfile** field:

   Sample IngressController CR for a Custom profile

   ```
   apiVersion: operator.openshift.io/v1
   ```

```
kind: IngressController
 ...
spec:
 tlsSecurityProfile:
  type: Custom ❶
  custom: ❷
   ciphers: ❸
   - ECDHE-ECDSA-CHACHA20-POLY1305
   - ECDHE-RSA-CHACHA20-POLY1305
   - ECDHE-RSA-AES128-GCM-SHA256
   - ECDHE-ECDSA-AES128-GCM-SHA256
   minTLSVersion: VersionTLS11
 ...
```

❶ Specify the TLS security profile type (**Old**, **Intermediate**, or **Custom**). The default is **Intermediate**.

❷ Specify the appropriate field for the selected type:

- **old: {}**

- **intermediate: {}**

- **custom:**

❸ For the **custom** type, specify a list of TLS ciphers and minimum accepted TLS version.

3. Save the file to apply the changes.

Verification

- Verify that the profile is set in the **IngressController** CR:

  ```
  $ oc describe IngressController default -n openshift-ingress-operator
  ```

Example output

```
Name:      default
Namespace:   openshift-ingress-operator
Labels:     <none>
Annotations: <none>
API Version: operator.openshift.io/v1
Kind:      IngressController
 ...
Spec:
 ...
  Tls Security Profile:
   Custom:
    Ciphers:
     ECDHE-ECDSA-CHACHA20-POLY1305
     ECDHE-RSA-CHACHA20-POLY1305
     ECDHE-RSA-AES128-GCM-SHA256
     ECDHE-ECDSA-AES128-GCM-SHA256
```

```
        Min TLS Version:  VersionTLS11
    Type:          Custom
  ...
```

7.3.1.3. Configuring mutual TLS authentication

You can configure the Ingress Controller to enable mutual TLS (mTLS) authentication by setting a **spec.clientTLS** value. The **clientTLS** value configures the Ingress Controller to verify client certificates. This configuration includes setting a **clientCA** value, which is a reference to a config map. The config map contains the PEM-encoded CA certificate bundle that is used to verify a client's certificate. Optionally, you can also configure a list of certificate subject filters.

If the **clientCA** value specifies an X509v3 certificate revocation list (CRL) distribution point, the Ingress Operator downloads and manages a CRL config map based on the HTTP URI X509v3 **CRL Distribution Point** specified in each provided certificate. The Ingress Controller uses this config map during mTLS/TLS negotiation. Requests that do not provide valid certificates are rejected.

Prerequisites

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

- You have a PEM-encoded CA certificate bundle.

- If your CA bundle references a CRL distribution point, you must have also included the end-entity or leaf certificate to the client CA bundle. This certificate must have included an HTTP URI under **CRL Distribution Points**, as described in RFC 5280. For example:

    ```
    Issuer: C=US, O=Example Inc, CN=Example Global G2 TLS RSA SHA256 2020 CA1
        Subject: SOME SIGNED CERT        X509v3 CRL Distribution Points:
            Full Name:
              URI:http://crl.example.com/example.crl
    ```

Procedure

1. In the **openshift-config** namespace, create a config map from your CA bundle:

    ```
    $ oc create configmap \
      router-ca-certs-default \
      --from-file=ca-bundle.pem=client-ca.crt \  ❶
      -n openshift-config
    ```

    ❶ The config map data key must be **ca-bundle.pem**, and the data value must be a CA certificate in PEM format.

2. Edit the **IngressController** resource in the **openshift-ingress-operator** project:

    ```
    $ oc edit IngressController default -n openshift-ingress-operator
    ```

3. Add the **spec.clientTLS** field and subfields to configure mutual TLS:

    Sample IngressController CR for a clientTLS profile that specifies filtering patterns

    ```
    apiVersion: operator.openshift.io/v1
    ```

```
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  clientTLS:
    clientCertificatePolicy: Required
    clientCA:
      name: router-ca-certs-default
    allowedSubjectPatterns:
    - "^/CN=example.com/ST=NC/C=US/O=Security/OU=OpenShift$"
```

7.4. VIEW THE DEFAULT INGRESS CONTROLLER

The Ingress Operator is a core feature of OpenShift Container Platform and is enabled out of the box.

Every new OpenShift Container Platform installation has an **ingresscontroller** named default. It can be supplemented with additional Ingress Controllers. If the default **ingresscontroller** is deleted, the Ingress Operator will automatically recreate it within a minute.

Procedure

- View the default Ingress Controller:

    ```
    $ oc describe --namespace=openshift-ingress-operator ingresscontroller/default
    ```

7.5. VIEW INGRESS OPERATOR STATUS

You can view and inspect the status of your Ingress Operator.

Procedure

- View your Ingress Operator status:

    ```
    $ oc describe clusteroperators/ingress
    ```

7.6. VIEW INGRESS CONTROLLER LOGS

You can view your Ingress Controller logs.

Procedure

- View your Ingress Controller logs:

    ```
    $ oc logs --namespace=openshift-ingress-operator deployments/ingress-operator -c
    <container_name>
    ```

7.7. VIEW INGRESS CONTROLLER STATUS

Your can view the status of a particular Ingress Controller.

Procedure

- View the status of an Ingress Controller:

  ```
  $ oc describe --namespace=openshift-ingress-operator ingresscontroller/<name>
  ```

7.8. CONFIGURING THE INGRESS CONTROLLER

7.8.1. Setting a custom default certificate

As an administrator, you can configure an Ingress Controller to use a custom certificate by creating a Secret resource and editing the **IngressController** custom resource (CR).

Prerequisites

- You must have a certificate/key pair in PEM-encoded files, where the certificate is signed by a trusted certificate authority or by a private trusted certificate authority that you configured in a custom PKI.

- Your certificate meets the following requirements:

  - The certificate is valid for the ingress domain.

  - The certificate uses the **subjectAltName** extension to specify a wildcard domain, such as ***.apps.ocp4.example.com**.

- You must have an **IngressController** CR. You may use the default one:

  ```
  $ oc --namespace openshift-ingress-operator get ingresscontrollers
  ```

  Example output

  ```
  NAME      AGE
  default   10m
  ```

> **NOTE**
>
> If you have intermediate certificates, they must be included in the **tls.crt** file of the secret containing a custom default certificate. Order matters when specifying a certificate; list your intermediate certificate(s) after any server certificate(s).

Procedure

The following assumes that the custom certificate and key pair are in the **tls.crt** and **tls.key** files in the current working directory. Substitute the actual path names for **tls.crt** and **tls.key**. You also may substitute another name for **custom-certs-default** when creating the Secret resource and referencing it in the IngressController CR.

> **NOTE**
>
> This action will cause the Ingress Controller to be redeployed, using a rolling deployment strategy.

1. Create a Secret resource containing the custom certificate in the **openshift-ingress** namespace using the **tls.crt** and **tls.key** files.

   ```
   $ oc --namespace openshift-ingress create secret tls custom-certs-default --cert=tls.crt --
   key=tls.key
   ```

2. Update the IngressController CR to reference the new certificate secret:

   ```
   $ oc patch --type=merge --namespace openshift-ingress-operator ingresscontrollers/default \
     --patch '{"spec":{"defaultCertificate":{"name":"custom-certs-default"}}}'
   ```

3. Verify the update was effective:

   ```
   $ echo Q |\
     openssl s_client -connect console-openshift-console.apps.<domain>:443 -showcerts
   2>/dev/null |\
     openssl x509 -noout -subject -issuer -enddate
   ```

   where:

   <domain>

   Specifies the base domain name for your cluster.

   Example output

   ```
   subject=C = US, ST = NC, L = Raleigh, O = RH, OU = OCP4, CN = *.apps.example.com
   issuer=C = US, ST = NC, L = Raleigh, O = RH, OU = OCP4, CN = example.com
   notAfter=May 10 08:32:45 2022 GM
   ```

   TIP

   You can alternatively apply the following YAML to set a custom default certificate:

   ```yaml
   apiVersion: operator.openshift.io/v1
   kind: IngressController
   metadata:
     name: default
     namespace: openshift-ingress-operator
   spec:
     defaultCertificate:
       name: custom-certs-default
   ```

   The certificate secret name should match the value used to update the CR.

Once the IngressController CR has been modified, the Ingress Operator updates the Ingress Controller's deployment to use the custom certificate.

7.8.2. Removing a custom default certificate

As an administrator, you can remove a custom certificate that you configured an Ingress Controller to use.

Prerequisites

Prerequisites

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

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

- You previously configured a custom default certificate for the Ingress Controller.

Procedure

- To remove the custom certificate and restore the certificate that ships with OpenShift Container Platform, enter the following command:

```
$ oc patch -n openshift-ingress-operator ingresscontrollers/default \
  --type json -p $'- op: remove\n  path: /spec/defaultCertificate'
```

There can be a delay while the cluster reconciles the new certificate configuration.

Verification

- To confirm that the original cluster certificate is restored, enter the following command:

```
$ echo Q | \
  openssl s_client -connect console-openshift-console.apps.<domain>:443 -showcerts
2>/dev/null | \
  openssl x509 -noout -subject -issuer -enddate
```

where:

<domain>

Specifies the base domain name for your cluster.

Example output

```
subject=CN = *.apps.<domain>
issuer=CN = ingress-operator@1620633373
notAfter=May 10 10:44:36 2023 GMT
```

7.8.3. Autoscaling an Ingress Controller

Automatically scale an Ingress Controller to dynamically meet routing performance or availability requirements such as the requirement to increase throughput. The following procedure provides an example for scaling up the default **IngressController**.

Prerequisites

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

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

3. You have the Custom Metrics Autoscaler Operator installed.

4. You are in the **openshift-ingress-operator** project namespace.

Procedure

1. Create a service account to authenticate with Thanos by running the following command:

   ```
   $ oc create serviceaccount thanos && oc describe serviceaccount thanos
   ```

   Example output

   ```
   Name:                thanos
   Namespace:           openshift-ingress-operator
   Labels:              <none>
   Annotations:         <none>
   Image pull secrets:  thanos-dockercfg-b4l9s
   Mountable secrets:   thanos-dockercfg-b4l9s
   Tokens:              thanos-token-c422q
   Events:              <none>
   ```

2. Define a **TriggerAuthentication** object within the **openshift-ingress-operator** namespace using the service account's token.

   a. Define the variable **secret** that contains the secret by running the following command:

   ```
   $ secret=$(oc get secret | grep thanos-token | head -n 1 | awk '{ print $1 }')
   ```

   b. Create the **TriggerAuthentication** object and pass the value of the **secret** variable to the **TOKEN** parameter:

   ```
   $ oc process TOKEN="$secret" -f - <<EOF | oc apply -f -
   apiVersion: template.openshift.io/v1
   kind: Template
   parameters:
   - name: TOKEN
   objects:
   - apiVersion: keda.sh/v1alpha1
     kind: TriggerAuthentication
     metadata:
       name: keda-trigger-auth-prometheus
     spec:
       secretTargetRef:
       - parameter: bearerToken
         name: \${TOKEN}
         key: token
       - parameter: ca
         name: \${TOKEN}
         key: ca.crt
   EOF
   ```

3. Create and apply a role for reading metrics from Thanos:

   a. Create a new role, **thanos-metrics-reader.yaml**, that reads metrics from pods and nodes:

   thanos-metrics-reader.yaml

   ```
   apiVersion: rbac.authorization.k8s.io/v1
   kind: Role
   ```

```
metadata:
  name: thanos-metrics-reader
rules:
- apiGroups:
  - ""
  resources:
  - pods
  - nodes
  verbs:
  - get
- apiGroups:
  - metrics.k8s.io
  resources:
  - pods
  - nodes
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - ""
  resources:
  - namespaces
  verbs:
  - get
```

b. Apply the new role by running the following command:

```
$ oc apply -f thanos-metrics-reader.yaml
```

4. Add the new role to the service account by entering the following commands:

```
$ oc adm policy add-role-to-user thanos-metrics-reader -z thanos --
role=namespace=openshift-ingress-operator
```

```
$ oc adm policy -n openshift-ingress-operator add-cluster-role-to-user cluster-monitoring-view
-z thanos
```
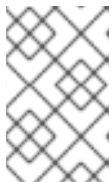
> **NOTE**
>
> The argument **add-cluster-role-to-user** is only required if you use cross-
> namespace queries. The following step uses a query from the **kube-metrics**
> namespace which requires this argument.

5. Create a new **ScaledObject** YAML file, **ingress-autoscaler.yaml**, that targets the default
Ingress Controller deployment:

Example ScaledObject definition

```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: ingress-scaler
```

```
spec:
  scaleTargetRef: 1
    apiVersion: operator.openshift.io/v1
    kind: IngressController
    name: default
    envSourceContainerName: ingress-operator
  minReplicaCount: 1
  maxReplicaCount: 20 2
  cooldownPeriod: 1
  pollingInterval: 1
  triggers:
  - type: prometheus
    metricType: AverageValue
    metadata:
      serverAddress: https://thanos-querier.openshift-monitoring.svc.cluster.local:9091 3
      namespace: openshift-ingress-operator 4
      metricName: 'kube-node-role'
      threshold: '1'
      query: 'sum(kube_node_role{role="worker",service="kube-state-metrics"})' 5
      authModes: "bearer"
    authenticationRef:
      name: keda-trigger-auth-prometheus
```

1 The custom resource that you are targeting. In this case, the Ingress Controller.

2 Optional: The maximum number of replicas. If you omit this field, the default maximum is set to 100 replicas.

3 The Thanos service endpoint in the **openshift-monitoring** namespace.

4 The Ingress Operator namespace.

5 This expression evaluates to however many worker nodes are present in the deployed cluster.

> **IMPORTANT**
>
> If you are using cross-namespace queries, you must target port 9091 and not port 9092 in the **serverAddress** field. You also must have elevated privileges to read metrics from this port.

6. Apply the custom resource definition by running the following command:

```
$ oc apply -f ingress-autoscaler.yaml
```

Verification

- Verify that the default Ingress Controller is scaled out to match the value returned by the **kube-state-metrics** query by running the following commands:

  - Use the **grep** command to search the Ingress Controller YAML file for replicas:

    ```
    $ oc get ingresscontroller/default -o yaml | grep replicas:
    ```

Example output

```
replicas: 3
```

- Get the pods in the **openshift-ingress** project:

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

Example output

```
NAME                          READY   STATUS    RESTARTS   AGE
router-default-7b5df44ff-l9pmm   2/2     Running   0          17h
router-default-7b5df44ff-s5sl5   2/2     Running   0          3d22h
router-default-7b5df44ff-wwsth   2/2     Running   0          66s
```

Additional resources

- Enabling monitoring for user-defined projects

- Installing the custom metrics autoscaler

- Understanding custom metrics autoscaler trigger authentications

- Configuring the custom metrics autoscaler to use OpenShift Container Platform monitoring

- Understanding how to add custom metrics autoscalers

7.8.4. Scaling an Ingress Controller

Manually scale an Ingress Controller to meeting routing performance or availability requirements such as the requirement to increase throughput. **oc** commands are used to scale the **IngressController** resource. The following procedure provides an example for scaling up the default **IngressController**.

> **NOTE**
>
> Scaling is not an immediate action, as it takes time to create the desired number of replicas.

Procedure

1. View the current number of available replicas for the default **IngressController**:

```
$ oc get -n openshift-ingress-operator ingresscontrollers/default -o
jsonpath='{$.status.availableReplicas}'
```

Example output

```
2
```

2. Scale the default **IngressController** to the desired number of replicas using the **oc patch** command. The following example scales the default **IngressController** to 3 replicas:

```
$ oc patch -n openshift-ingress-operator ingresscontroller/default --patch '{"spec":{"replicas":
3}}' --type=merge
```

Example output

```
ingresscontroller.operator.openshift.io/default patched
```

3. Verify that the default **IngressController** scaled to the number of replicas that you specified:

```
$ oc get -n openshift-ingress-operator ingresscontrollers/default -o
jsonpath='{$.status.availableReplicas}'
```

Example output

```
3
```

TIP

You can alternatively apply the following YAML to scale an Ingress Controller to three replicas:

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  replicas: 3          1
```

[1]   If you need a different amount of replicas, change the **replicas** value.

7.8.5. Configuring Ingress access logging

You can configure the Ingress Controller to enable access logs. If you have clusters that do not receive much traffic, then you can log to a sidecar. If you have high traffic clusters, to avoid exceeding the capacity of the logging stack or to integrate with a logging infrastructure outside of OpenShift Container Platform, you can forward logs to a custom syslog endpoint. You can also specify the format for access logs.

Container logging is useful to enable access logs on low-traffic clusters when there is no existing Syslog logging infrastructure, or for short-term use while diagnosing problems with the Ingress Controller.

Syslog is needed for high-traffic clusters where access logs could exceed the OpenShift Logging stack's capacity, or for environments where any logging solution needs to integrate with an existing Syslog logging infrastructure. The Syslog use-cases can overlap.

Prerequisites

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

Procedure

Configure Ingress access logging to a sidecar.

- To configure Ingress access logging, you must specify a destination using **spec.logging.access.destination**. To specify logging to a sidecar container, you must specify **Container spec.logging.access.destination.type**. The following example is an Ingress Controller definition that logs to a **Container** destination:

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  replicas: 2
  logging:
    access:
      destination:
        type: Container
```

- When you configure the Ingress Controller to log to a sidecar, the operator creates a container named **logs** inside the Ingress Controller Pod:

```
$ oc -n openshift-ingress logs deployment.apps/router-default -c logs
```

Example output

```
2020-05-11T19:11:50.135710+00:00 router-default-57dfc6cd95-bpmk6 router-default-
57dfc6cd95-bpmk6 haproxy[108]: 174.19.21.82:39654 [11/May/2020:19:11:50.133] public
be_http:hello-openshift:hello-openshift/pod:hello-openshift:hello-openshift:10.128.2.12:8080
0/0/1/0/1 200 142 - - --NI 1/1/0/0/0 0/0 "GET / HTTP/1.1"
```

Configure Ingress access logging to a Syslog endpoint.

- To configure Ingress access logging, you must specify a destination using **spec.logging.access.destination**. To specify logging to a Syslog endpoint destination, you must specify **Syslog** for **spec.logging.access.destination.type**. If the destination type is **Syslog**, you must also specify a destination endpoint using **spec.logging.access.destination.syslog.endpoint** and you can specify a facility using **spec.logging.access.destination.syslog.facility**. The following example is an Ingress Controller definition that logs to a **Syslog** destination:

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  replicas: 2
  logging:
    access:
      destination:
        type: Syslog
        syslog:
          address: 1.2.3.4
          port: 10514
```

> **NOTE**
>
> The **syslog** destination port must be UDP.

Configure Ingress access logging with a specific log format.

- You can specify **spec.logging.access.httpLogFormat** to customize the log format. The following example is an Ingress Controller definition that logs to a **syslog** endpoint with IP address 1.2.3.4 and port 10514:

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  replicas: 2
  logging:
    access:
      destination:
        type: Syslog
        syslog:
          address: 1.2.3.4
          port: 10514
      httpLogFormat: '%ci:%cp [%t] %ft %b/%s %B %bq %HM %HU %HV'
```

Disable Ingress access logging.

- To disable Ingress access logging, leave **spec.logging** or **spec.logging.access** empty:

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  replicas: 2
  logging:
    access: null
```

7.8.6. Setting Ingress Controller thread count

A cluster administrator can set the thread count to increase the amount of incoming connections a cluster can handle. You can patch an existing Ingress Controller to increase the amount of threads.

Prerequisites

- The following assumes that you already created an Ingress Controller.

Procedure

- Update the Ingress Controller to increase the number of threads:

```
$ oc -n openshift-ingress-operator patch ingresscontroller/default --type=merge -p '{"spec":
{"tuningOptions": {"threadCount": 8}}}'
```
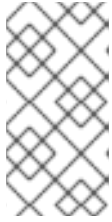
NOTE

If you have a node that is capable of running large amounts of resources, you can configure **spec.nodePlacement.nodeSelector** with labels that match the capacity of the intended node, and configure **spec.tuningOptions.threadCount** to an appropriately high value.

7.8.7. Configuring an Ingress Controller to use an internal load balancer

When creating an Ingress Controller on cloud platforms, the Ingress Controller is published by a public cloud load balancer by default. As an administrator, you can create an Ingress Controller that uses an internal cloud load balancer.

WARNING

If your cloud provider is Microsoft Azure, you must have at least one public load balancer that points to your nodes. If you do not, all of your nodes will lose egress connectivity to the internet.
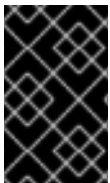
IMPORTANT

If you want to change the **scope** for an **IngressController**, you can change the **.spec.endpointPublishingStrategy.loadBalancer.scope** parameter after the custom resource (CR) is created.

Figure 7.1. Diagram of LoadBalancer
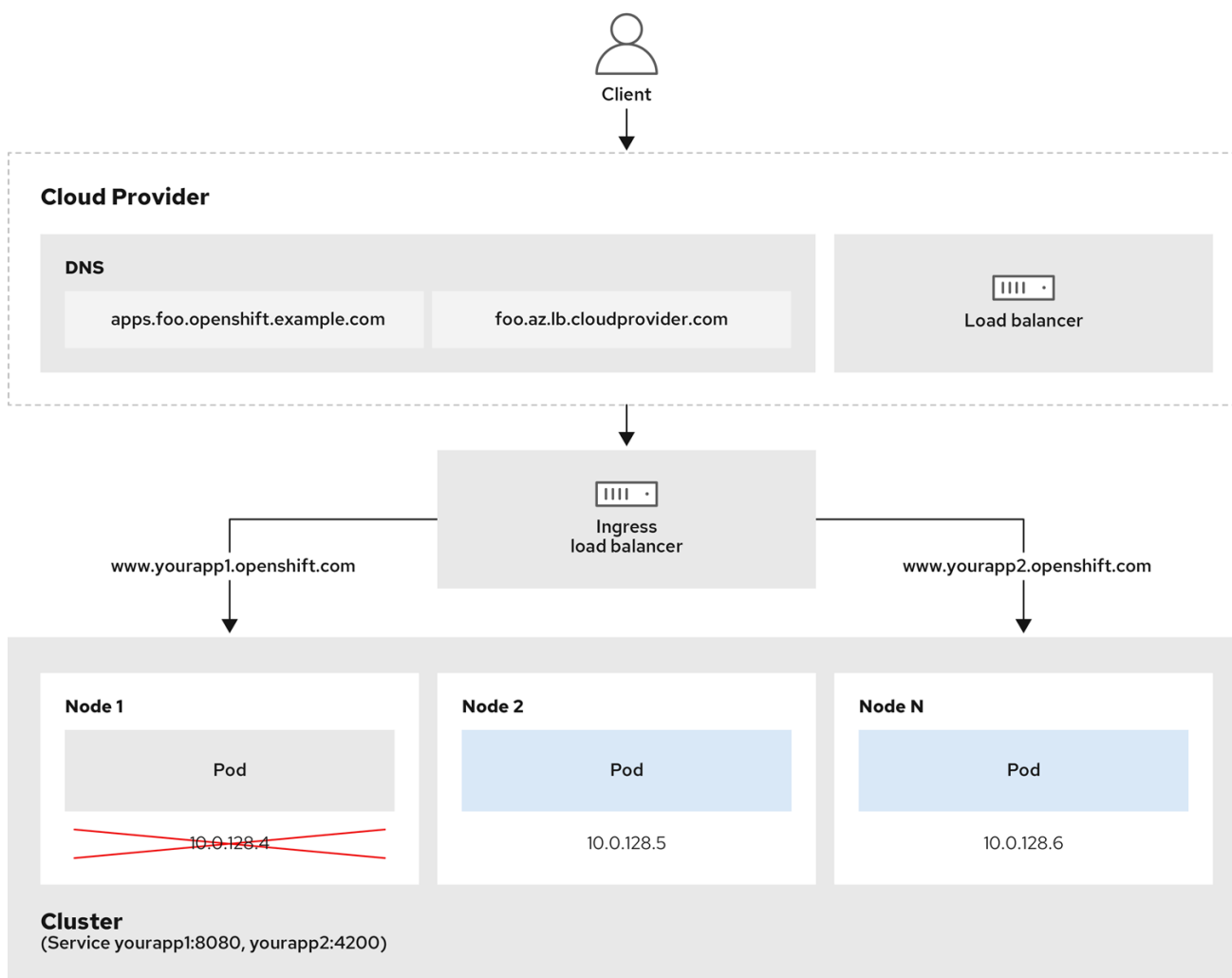
The preceding graphic shows the following concepts pertaining to OpenShift Container Platform Ingress LoadBalancerService endpoint publishing strategy:

- You can load balance externally, using the cloud provider load balancer, or internally, using the OpenShift Ingress Controller Load Balancer.

- You can use the single IP address of the load balancer and more familiar ports, such as 8080 and 4200 as shown on the cluster depicted in the graphic.

- Traffic from the external load balancer is directed at the pods, and managed by the load balancer, as depicted in the instance of a down node. See the Kubernetes Services documentation for implementation details.

Prerequisites

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

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

Procedure

1. Create an **IngressController** custom resource (CR) in a file named **<name>-ingress-controller.yaml**, such as in the following example:

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  namespace: openshift-ingress-operator
  name: <name> ❶
spec:
  domain: <domain> ❷
  endpointPublishingStrategy:
    type: LoadBalancerService
    loadBalancer:
      scope: Internal ❸
```

❶ Replace **<name>** with a name for the **IngressController** object.

❷ Specify the **domain** for the application published by the controller.

❸ Specify a value of **Internal** to use an internal load balancer.

2. Create the Ingress Controller defined in the previous step by running the following command:

```
$ oc create -f <name>-ingress-controller.yaml ❶
```

❶ Replace **<name>** with the name of the **IngressController** object.

3. Optional: Confirm that the Ingress Controller was created by running the following command:

```
$ oc --all-namespaces=true get ingresscontrollers
```

7.8.8. Configuring global access for an Ingress Controller on GCP

An Ingress Controller created on GCP with an internal load balancer generates an internal IP address for the service. A cluster administrator can specify the global access option, which enables clients in any region within the same VPC network and compute region as the load balancer, to reach the workloads running on your cluster.

For more information, see the GCP documentation for global access.

Prerequisites

- You deployed an OpenShift Container Platform cluster on GCP infrastructure.

- You configured an Ingress Controller to use an internal load balancer.

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

Procedure

1. Configure the Ingress Controller resource to allow global access.

  NOTE

  You can also create an Ingress Controller and specify the global access option.

a. Configure the Ingress Controller resource:

```
$ oc -n openshift-ingress-operator edit ingresscontroller/default
```

b. Edit the YAML file:

Sample clientAccess configuration to Global

```
spec:
  endpointPublishingStrategy:
    loadBalancer:
      providerParameters:
        gcp:
          clientAccess: Global 1
        type: GCP
      scope: Internal
    type: LoadBalancerService
```

1. Set **gcp.clientAccess** to **Global**.

c. Save the file to apply the changes.

2. Run the following command to verify that the service allows global access:

```
$ oc -n openshift-ingress edit svc/router-default -o yaml
```

The output shows that global access is enabled for GCP with the annotation, **networking.gke.io/internal-load-balancer-allow-global-access**.

7.8.9. Setting the Ingress Controller health check interval

A cluster administrator can set the health check interval to define how long the router waits between two consecutive health checks. This value is applied globally as a default for all routes. The default value is 5 seconds.

Prerequisites

- The following assumes that you already created an Ingress Controller.

Procedure

- Update the Ingress Controller to change the interval between back end health checks:

```
$ oc -n openshift-ingress-operator patch ingresscontroller/default --type=merge -p '{"spec":
{"tuningOptions": {"healthCheckInterval": "8s"}}}'
```

> **NOTE**
>
> To override the **healthCheckInterval** for a single route, use the route annotation **router.openshift.io/haproxy.health.check.interval**

7.8.10. Configuring the default Ingress Controller for your cluster to be internal

You can configure the **default** Ingress Controller for your cluster to be internal by deleting and recreating it.

> **WARNING**
>
> If your cloud provider is Microsoft Azure, you must have at least one public load balancer that points to your nodes. If you do not, all of your nodes will lose egress connectivity to the internet.
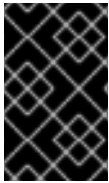
> **IMPORTANT**
>
> If you want to change the **scope** for an **IngressController**, you can change the **.spec.endpointPublishingStrategy.loadBalancer.scope** parameter after the custom resource (CR) is created.

Prerequisites

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

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

Procedure

1. Configure the **default** Ingress Controller for your cluster to be internal by deleting and recreating it.

   ```
   $ oc replace --force --wait --filename - <<EOF
   apiVersion: operator.openshift.io/v1
   kind: IngressController
   metadata:
     namespace: openshift-ingress-operator
     name: default
   spec:
     endpointPublishingStrategy:
       type: LoadBalancerService
       loadBalancer:
         scope: Internal
   EOF
   ```

7.8.11. Configuring the route admission policy

Administrators and application developers can run applications in multiple namespaces with the same domain name. This is for organizations where multiple teams develop microservices that are exposed on the same hostname.

> **WARNING**
>
> Allowing claims across namespaces should only be enabled for clusters with trust between namespaces, otherwise a malicious user could take over a hostname. For this reason, the default admission policy disallows hostname claims across namespaces.

Prerequisites

- Cluster administrator privileges.

Procedure

- Edit the **.spec.routeAdmission** field of the **ingresscontroller** resource variable using the following command:

  ```
  $ oc -n openshift-ingress-operator patch ingresscontroller/default --patch '{"spec":
  {"routeAdmission":{"namespaceOwnership":"InterNamespaceAllowed"}}}' --type=merge
  ```

  Sample Ingress Controller configuration

  ```
  spec:
    routeAdmission:
      namespaceOwnership: InterNamespaceAllowed
  ...
  ```

  TIP

  You can alternatively apply the following YAML to configure the route admission policy:

  ```
  apiVersion: operator.openshift.io/v1
  kind: IngressController
  metadata:
    name: default
    namespace: openshift-ingress-operator
  spec:
    routeAdmission:
      namespaceOwnership: InterNamespaceAllowed
  ```

7.8.12. Using wildcard routes

The HAProxy Ingress Controller has support for wildcard routes. The Ingress Operator uses **wildcardPolicy** to configure the **ROUTER_ALLOW_WILDCARD_ROUTES** environment variable of the Ingress Controller.

The default behavior of the Ingress Controller is to admit routes with a wildcard policy of **None**, which is backwards compatible with existing **IngressController** resources.

Procedure

1. Configure the wildcard policy.

    a. Use the following command to edit the **IngressController** resource:

    ```
    $ oc edit IngressController
    ```

    b. Under **spec**, set the **wildcardPolicy** field to **WildcardsDisallowed** or **WildcardsAllowed**:

    ```
    spec:
      routeAdmission:
        wildcardPolicy: WildcardsDisallowed # or WildcardsAllowed
    ```

7.8.13. Using X-Forwarded headers

You configure the HAProxy Ingress Controller to specify a policy for how to handle HTTP headers including **Forwarded** and **X-Forwarded-For**. The Ingress Operator uses the **HTTPHeaders** field to configure the **ROUTER_SET_FORWARDED_HEADERS** environment variable of the Ingress Controller.

Procedure

1. Configure the **HTTPHeaders** field for the Ingress Controller.

    a. Use the following command to edit the **IngressController** resource:

    ```
    $ oc edit IngressController
    ```

    b. Under **spec**, set the **HTTPHeaders** policy field to **Append**, **Replace**, **IfNone**, or **Never**:

    ```
    apiVersion: operator.openshift.io/v1
    kind: IngressController
    metadata:
      name: default
      namespace: openshift-ingress-operator
    spec:
      httpHeaders:
        forwardedHeaderPolicy: Append
    ```

Example use cases
As a cluster administrator, you can:

- Configure an external proxy that injects the **X-Forwarded-For** header into each request before forwarding it to an Ingress Controller.
  To configure the Ingress Controller to pass the header through unmodified, you specify the **never** policy. The Ingress Controller then never sets the headers, and applications receive only the headers that the external proxy provides.

- Configure the Ingress Controller to pass the **X-Forwarded-For** header that your external proxy sets on external cluster requests through unmodified.
  To configure the Ingress Controller to set the **X-Forwarded-For** header on internal cluster requests, which do not go through the external proxy, specify the **if-none** policy. If an HTTP request already has the header set through the external proxy, then the Ingress Controller preserves it. If the header is absent because the request did not come through the proxy, then the Ingress Controller adds the header.

As an application developer, you can:

- Configure an application-specific external proxy that injects the **X-Forwarded-For** header. To configure an Ingress Controller to pass the header through unmodified for an application's Route, without affecting the policy for other Routes, add an annotation **haproxy.router.openshift.io/set-forwarded-headers: if-none** or **haproxy.router.openshift.io/set-forwarded-headers: never** on the Route for the application.

> **NOTE**
>
> You can set the **haproxy.router.openshift.io/set-forwarded-headers** annotation on a per route basis, independent from the globally set value for the Ingress Controller.

7.8.14. Enabling HTTP/2 Ingress connectivity

You can enable transparent end-to-end HTTP/2 connectivity in HAProxy. It allows application owners to make use of HTTP/2 protocol capabilities, including single connection, header compression, binary streams, and more.

You can enable HTTP/2 connectivity for an individual Ingress Controller or for the entire cluster.

To enable the use of HTTP/2 for the connection from the client to HAProxy, a route must specify a custom certificate. A route that uses the default certificate cannot use HTTP/2. This restriction is necessary to avoid problems from connection coalescing, where the client re-uses a connection for different routes that use the same certificate.

The connection from HAProxy to the application pod can use HTTP/2 only for re-encrypt routes and not for edge-terminated or insecure routes. This restriction is because HAProxy uses Application-Level Protocol Negotiation (ALPN), which is a TLS extension, to negotiate the use of HTTP/2 with the back-end. The implication is that end-to-end HTTP/2 is possible with passthrough and re-encrypt and not with insecure or edge-terminated routes.

> **WARNING**
>
> Using WebSockets with a re-encrypt route and with HTTP/2 enabled on an Ingress Controller requires WebSocket support over HTTP/2. WebSockets over HTTP/2 is a feature of HAProxy 2.4, which is unsupported in OpenShift Container Platform at this time.
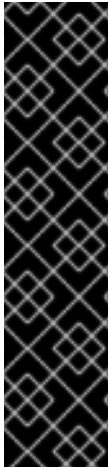
IMPORTANT

For non-passthrough routes, the Ingress Controller negotiates its connection to the application independently of the connection from the client. This means a client may connect to the Ingress Controller and negotiate HTTP/1.1, and the Ingress Controller may then connect to the application, negotiate HTTP/2, and forward the request from the client HTTP/1.1 connection using the HTTP/2 connection to the application. This poses a problem if the client subsequently tries to upgrade its connection from HTTP/1.1 to the WebSocket protocol, because the Ingress Controller cannot forward WebSocket to HTTP/2 and cannot upgrade its HTTP/2 connection to WebSocket. Consequently, if you have an application that is intended to accept WebSocket connections, it must not allow negotiating the HTTP/2 protocol or else clients will fail to upgrade to the WebSocket protocol.

Procedure

Enable HTTP/2 on a single Ingress Controller.

- To enable HTTP/2 on an Ingress Controller, enter the **oc annotate** command:

  ```
  $ oc -n openshift-ingress-operator annotate ingresscontrollers/<ingresscontroller_name>
  ingress.operator.openshift.io/default-enable-http2=true
  ```

  Replace **<ingresscontroller_name>** with the name of the Ingress Controller to annotate.

Enable HTTP/2 on the entire cluster.

- To enable HTTP/2 for the entire cluster, enter the **oc annotate** command:

  ```
  $ oc annotate ingresses.config/cluster ingress.operator.openshift.io/default-enable-http2=true
  ```

  TIP

  You can alternatively apply the following YAML to add the annotation:

  ```
  apiVersion: config.openshift.io/v1
  kind: Ingress
  metadata:
    name: cluster
    annotations:
      ingress.operator.openshift.io/default-enable-http2: "true"
  ```

7.8.15. Configuring the PROXY protocol for an Ingress Controller

A cluster administrator can configure the PROXY protocol when an Ingress Controller uses either the **HostNetwork** or **NodePortService** endpoint publishing strategy types. The PROXY protocol enables the load balancer to preserve the original client addresses for connections that the Ingress Controller receives. The original client addresses are useful for logging, filtering, and injecting HTTP headers. In the default configuration, the connections that the Ingress Controller receives only contain the source address that is associated with the load balancer.

This feature is not supported in cloud deployments. This restriction is because when OpenShift Container Platform runs in a cloud platform, and an IngressController specifies that a service load balancer should be used, the Ingress Operator configures the load balancer service and enables the

PROXY protocol based on the platform requirement for preserving source addresses.

> **IMPORTANT**
>
> You must configure both OpenShift Container Platform and the external load balancer to either use the PROXY protocol or to use TCP.

> **WARNING**
>
> The PROXY protocol is unsupported for the default Ingress Controller with installer-provisioned clusters on non-cloud platforms that use a Keepalived Ingress VIP.

Prerequisites

- You created an Ingress Controller.

Procedure

1. Edit the Ingress Controller resource:

   ```
   $ oc -n openshift-ingress-operator edit ingresscontroller/default
   ```

2. Set the PROXY configuration:

   - If your Ingress Controller uses the hostNetwork endpoint publishing strategy type, set the **spec.endpointPublishingStrategy.hostNetwork.protocol** subfield to **PROXY**:

     Sample **hostNetwork** configuration to **PROXY**

     ```
     spec:
       endpointPublishingStrategy:
         hostNetwork:
           protocol: PROXY
         type: HostNetwork
     ```

   - If your Ingress Controller uses the NodePortService endpoint publishing strategy type, set the **spec.endpointPublishingStrategy.nodePort.protocol** subfield to **PROXY**:

     Sample **nodePort** configuration to **PROXY**

     ```
     spec:
       endpointPublishingStrategy:
         nodePort:
           protocol: PROXY
         type: NodePortService
     ```

7.8.16. Specifying an alternative cluster domain using the appsDomain option

As a cluster administrator, you can specify an alternative to the default cluster domain for user-created routes by configuring the **appsDomain** field. The **appsDomain** field is an optional domain for OpenShift Container Platform to use instead of the default, which is specified in the **domain** field. If you specify an alternative domain, it overrides the default cluster domain for the purpose of determining the default host for a new route.

For example, you can use the DNS domain for your company as the default domain for routes and ingresses for applications running on your cluster.

Prerequisites

- You deployed an OpenShift Container Platform cluster.

- You installed the **oc** command line interface.

Procedure

1. Configure the **appsDomain** field by specifying an alternative default domain for user-created routes.

   a. Edit the ingress **cluster** resource:

   ```
   $ oc edit ingresses.config/cluster -o yaml
   ```

   b. Edit the YAML file:

   Sample appsDomain configuration to test.example.com

   ```
   apiVersion: config.openshift.io/v1
   kind: Ingress
   metadata:
     name: cluster
   spec:
     domain: apps.example.com          1
     appsDomain: <test.example.com>    2
   ```

   1   Specifies the default domain. You cannot modify the default domain after installation.

   2   Optional: Domain for OpenShift Container Platform infrastructure to use for application routes. Instead of the default prefix, **apps**, you can use an alternative prefix like **test**.

2. Verify that an existing route contains the domain name specified in the **appsDomain** field by exposing the route and verifying the route domain change:

   > **NOTE**
   >
   > Wait for the **openshift-apiserver** finish rolling updates before exposing the route.

   a. Expose the route:

```
$ oc expose service hello-openshift
route.route.openshift.io/hello-openshift exposed
```

Example output:

```
$ oc get routes
NAME            HOST/PORT                                    PATH  SERVICES        PORT
TERMINATION  WILDCARD
hello-openshift   hello_openshift-<my_project>.test.example.com
hello-openshift   8080-tcp              None
```

7.8.17. Converting HTTP header case

HAProxy 2.2 lowercases HTTP header names by default, for example, changing **Host: xyz.com** to **host: xyz.com**. If legacy applications are sensitive to the capitalization of HTTP header names, use the Ingress Controller **spec.httpHeaders.headerNameCaseAdjustments** API field for a solution to accommodate legacy applications until they can be fixed.

> **IMPORTANT**
>
> Because OpenShift Container Platform includes HAProxy 2.2, make sure to add the necessary configuration by using **spec.httpHeaders.headerNameCaseAdjustments** before upgrading.

Prerequisites

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

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

Procedure

As a cluster administrator, you can convert the HTTP header case by entering the **oc patch** command or by setting the **HeaderNameCaseAdjustments** field in the Ingress Controller YAML file.

- Specify an HTTP header to be capitalized by entering the **oc patch** command.

    1. Enter the **oc patch** command to change the HTTP **host** header to **Host**:

        ```
        $ oc -n openshift-ingress-operator patch ingresscontrollers/default --type=merge --patch='{"spec":{"httpHeaders":{"headerNameCaseAdjustments":["Host"]}}}'
        ```

    2. Annotate the route of the application:

        ```
        $ oc annotate routes/my-application haproxy.router.openshift.io/h1-adjust-case=true
        ```

        The Ingress Controller then adjusts the **host** request header as specified.

- Specify adjustments using the **HeaderNameCaseAdjustments** field by configuring the Ingress Controller YAML file.

    1. The following example Ingress Controller YAML adjusts the **host** header to **Host** for HTTP/1 requests to appropriately annotated routes:

Example Ingress Controller YAML

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  httpHeaders:
    headerNameCaseAdjustments:
    - Host
```

2. The following example route enables HTTP response header name case adjustments using the **haproxy.router.openshift.io/h1-adjust-case** annotation:

Example route YAML

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  annotations:
    haproxy.router.openshift.io/h1-adjust-case: true  ❶
  name: my-application
  namespace: my-application
spec:
  to:
    kind: Service
    name: my-application
```

❶ Set **haproxy.router.openshift.io/h1-adjust-case** to true.

7.8.18. Using router compression

You configure the HAProxy Ingress Controller to specify router compression globally for specific MIME types. You can use the **mimeTypes** variable to define the formats of MIME types to which compression is applied. The types are: application, image, message, multipart, text, video, or a custom type prefaced by "X-". To see the full notation for MIME types and subtypes, see RFC1341.

> **NOTE**
>
> Memory allocated for compression can affect the max connections. Additionally, compression of large buffers can cause latency, like heavy regex or long lists of regex.
>
> Not all MIME types benefit from compression, but HAProxy still uses resources to try to compress if instructed to. Generally, text formats, such as html, css, and js, formats benefit from compression, but formats that are already compressed, such as image, audio, and video, benefit little in exchange for the time and resources spent on compression.

Procedure

1. Configure the **httpCompression** field for the Ingress Controller.

   a. Use the following command to edit the **IngressController** resource:

```
$ oc edit -n openshift-ingress-operator ingresscontrollers/default
```

b. Under **spec**, set the **httpCompression** policy field to **mimeTypes** and specify a list of MIME types that should have compression applied:

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  httpCompression:
    mimeTypes:
    - "text/html"
    - "text/css; charset=utf-8"
    - "application/json"
   ...
```

7.8.19. Exposing router metrics

You can expose the HAProxy router metrics by default in Prometheus format on the default stats port, 1936. The external metrics collection and aggregation systems such as Prometheus can access the HAProxy router metrics. You can view the HAProxy router metrics in a browser in the HTML and comma separated values (CSV) format.

Prerequisites

- You configured your firewall to access the default stats port, 1936.

Procedure

1. Get the router pod name by running the following command:

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

Example output

```
NAME                           READY  STATUS   RESTARTS  AGE
router-default-76bfffb66c-46qwp  1/1    Running  0         11h
```

2. Get the router's username and password, which the router pod stores in the **/var/lib/haproxy/conf/metrics-auth/statsUsername** and **/var/lib/haproxy/conf/metrics-auth/statsPassword** files:

   a. Get the username by running the following command:

   ```
   $ oc rsh <router_pod_name> cat metrics-auth/statsUsername
   ```

   b. Get the password by running the following command:

   ```
   $ oc rsh <router_pod_name> cat metrics-auth/statsPassword
   ```

3. Get the router IP and metrics certificates by running the following command:

```
$ oc describe pod <router_pod>
```

4. Get the raw statistics in Prometheus format by running the following command:

```
$ curl -u <user>:<password> http://<router_IP>:<stats_port>/metrics
```

5. Access the metrics securely by running the following command:

```
$ curl -u user:password https://<router_IP>:<stats_port>/metrics -k
```

6. Access the default stats port, 1936, by running the following command:

```
$ curl -u <user>:<password> http://<router_IP>:<stats_port>/metrics
```

Example 7.1. Example output

```
...
# HELP haproxy_backend_connections_total Total number of connections.
# TYPE haproxy_backend_connections_total gauge
haproxy_backend_connections_total{backend="http",namespace="default",route="hello-
route"} 0
haproxy_backend_connections_total{backend="http",namespace="default",route="hello-
route-alt"} 0
haproxy_backend_connections_total{backend="http",namespace="default",route="hello-
route01"} 0
...
# HELP haproxy_exporter_server_threshold Number of servers tracked and the current
threshold value.
# TYPE haproxy_exporter_server_threshold gauge
haproxy_exporter_server_threshold{type="current"} 11
haproxy_exporter_server_threshold{type="limit"} 500
...
# HELP haproxy_frontend_bytes_in_total Current total of incoming bytes.
# TYPE haproxy_frontend_bytes_in_total gauge
haproxy_frontend_bytes_in_total{frontend="fe_no_sni"} 0
haproxy_frontend_bytes_in_total{frontend="fe_sni"} 0
haproxy_frontend_bytes_in_total{frontend="public"} 119070
...
# HELP haproxy_server_bytes_in_total Current total of incoming bytes.
# TYPE haproxy_server_bytes_in_total gauge
haproxy_server_bytes_in_total{namespace="",pod="",route="",server="fe_no_sni",service="
"} 0
haproxy_server_bytes_in_total{namespace="",pod="",route="",server="fe_sni",service=""}
0
haproxy_server_bytes_in_total{namespace="default",pod="docker-registry-5-
nk5fz",route="docker-registry",server="10.130.0.89:5000",service="docker-registry"} 0
haproxy_server_bytes_in_total{namespace="default",pod="hello-rc-vkjqx",route="hello-
route",server="10.130.0.90:8080",service="hello-svc-1"} 0
...
```

7. Launch the stats window by entering the following URL in a browser:

> http://<user>:<password>@<router_IP>:<stats_port>

8. Optional: Get the stats in CSV format by entering the following URL in a browser:

> http://<user>:<password>@<router_ip>:1936/metrics;csv

7.8.20. Customizing HAProxy error code response pages

As a cluster administrator, you can specify a custom error code response page for either 503, 404, or both error pages. The HAProxy router serves a 503 error page when the application pod is not running or a 404 error page when the requested URL does not exist. For example, if you customize the 503 error code response page, then the page is served when the application pod is not running, and the default 404 error code HTTP response page is served by the HAProxy router for an incorrect route or a non-existing route.

Custom error code response pages are specified in a config map then patched to the Ingress Controller. The config map keys have two available file names as follows: **error-page-503.http** and **error-page-404.http**.

Custom HTTP error code response pages must follow the HAProxy HTTP error page configuration guidelines. Here is an example of the default OpenShift Container Platform HAProxy router http 503 error code response page. You can use the default content as a template for creating your own custom page.

By default, the HAProxy router serves only a 503 error page when the application is not running or when the route is incorrect or non-existent. This default behavior is the same as the behavior on OpenShift Container Platform 4.8 and earlier. If a config map for the customization of an HTTP error code response is not provided, and you are using a custom HTTP error code response page, the router serves a default 404 or 503 error code response page.

> **NOTE**
>
> If you use the OpenShift Container Platform default 503 error code page as a template for your customizations, the headers in the file require an editor that can use CRLF line endings.

Procedure

1. Create a config map named **my-custom-error-code-pages** in the **openshift-config** namespace:

   ```
   $ oc -n openshift-config create configmap my-custom-error-code-pages \
   --from-file=error-page-503.http \
   --from-file=error-page-404.http
   ```

   > **IMPORTANT**
   >
   > If you do not specify the correct format for the custom error code response page, a router pod outage occurs. To resolve this outage, you must delete or correct the config map and delete the affected router pods so they can be recreated with the correct information.

2. Patch the Ingress Controller to reference the **my-custom-error-code-pages** config map by name:

```
$ oc patch -n openshift-ingress-operator ingresscontroller/default --patch '{"spec":
{"httpErrorCodePages":{"name":"my-custom-error-code-pages"}}}' --type=merge
```

The Ingress Operator copies the **my-custom-error-code-pages** config map from the **openshift-config** namespace to the **openshift-ingress** namespace. The Operator names the config map according to the pattern, **<your_ingresscontroller_name>-errorpages**, in the **openshift-ingress** namespace.

3. Display the copy:

```
$ oc get cm default-errorpages -n openshift-ingress
```

Example output

```
NAME                DATA  AGE
default-errorpages    2    25s     1
```

1   The example config map name is **default-errorpages** because the **default** Ingress Controller custom resource (CR) was patched.

4. Confirm that the config map containing the custom error response page mounts on the router volume where the config map key is the filename that has the custom HTTP error code response:

- For 503 custom HTTP custom error code response:

```
$ oc -n openshift-ingress rsh <router_pod> cat
/var/lib/haproxy/conf/error_code_pages/error-page-503.http
```

- For 404 custom HTTP custom error code response:

```
$ oc -n openshift-ingress rsh <router_pod> cat
/var/lib/haproxy/conf/error_code_pages/error-page-404.http
```

Verification

Verify your custom error code HTTP response:

1. Create a test project and application:

```
$ oc new-project test-ingress
```

```
$ oc new-app django-psql-example
```

2. For 503 custom http error code response:

   a. Stop all the pods for the application.

   b. Run the following curl command or visit the route hostname in the browser:

```
$ curl -vk <route_hostname>
```

3. For 404 custom http error code response:

    a. Visit a non-existent route or an incorrect route.

    b. Run the following curl command or visit the route hostname in the browser:

    ```
    $ curl -vk <route_hostname>
    ```

4. Check if the **errorfile** attribute is properly in the **haproxy.config** file:

    ```
    $ oc -n openshift-ingress rsh <router> cat /var/lib/haproxy/conf/haproxy.config | grep errorfile
    ```

7.8.21. Setting the Ingress Controller maximum connections

A cluster administrator can set the maximum number of simultaneous connections for OpenShift router deployments. You can patch an existing Ingress Controller to increase the maximum number of connections.

Prerequisites

- The following assumes that you already created an Ingress Controller

Procedure

- Update the Ingress Controller to change the maximum number of connections for HAProxy:

    ```
    $ oc -n openshift-ingress-operator patch ingresscontroller/default --type=merge -p '{"spec": {"tuningOptions": {"maxConnections": 7500}}}'
    ```

> **WARNING**
>
> If you set the **spec.tuningOptions.maxConnections** value greater than the current operating system limit, the HAProxy process will not start. See the table in the "Ingress Controller configuration parameters" section for more information about this parameter.

7.9. ADDITIONAL RESOURCES

- Configuring a custom PKI

CHAPTER 8. INGRESS SHARDING IN OPENSHIFT CONTAINER PLATFORM

In OpenShift Container Platform, an Ingress Controller can serve all routes, or it can serve a subset of routes. By default, the Ingress Controller serves any route created in any namespace in the cluster. You can add additional Ingress Controllers to your cluster to optimize routing by creating *shards*, which are subsets of routes based on selected characteristics. To mark a route as a member of a shard, use labels in the route or namespace **metadata** field. The Ingress Controller uses *selectors*, also known as a *selection expression*, to select a subset of routes from the entire pool of routes to serve.

Ingress sharding is useful in cases where you want to load balance incoming traffic across multiple Ingress Controllers, when you want to isolate traffic to be routed to a specific Ingress Controller, or for a variety of other reasons described in the next section.

By default, each route uses the default domain of the cluster. However, routes can be configured to use the domain of the router instead. For more information, see Creating a route for Ingress Controller Sharding.

8.1. INGRESS CONTROLLER SHARDING

You can use Ingress sharding, also known as router sharding, to distribute a set of routes across multiple routers by adding labels to routes, namespaces, or both. The Ingress Controller uses a corresponding set of selectors to admit only the routes that have a specified label. Each Ingress shard comprises the routes that are filtered using a given selection expression.

As the primary mechanism for traffic to enter the cluster, the demands on the Ingress Controller can be significant. As a cluster administrator, you can shard the routes to:

- Balance Ingress Controllers, or routers, with several routes to speed up responses to changes.

- Allocate certain routes to have different reliability guarantees than other routes.

- Allow certain Ingress Controllers to have different policies defined.

- Allow only specific routes to use additional features.

- Expose different routes on different addresses so that internal and external users can see different routes, for example.

- Transfer traffic from one version of an application to another during a blue green deployment.

When Ingress Controllers are sharded, a given route is admitted to zero or more Ingress Controllers in the group. A route's status describes whether an Ingress Controller has admitted it or not. An Ingress Controller will only admit a route if it is unique to its shard.

An Ingress Controller can use three sharding methods:

- Adding only a namespace selector to the Ingress Controller, so that all routes in a namespace with labels that match the namespace selector are in the Ingress shard.

- Adding only a route selector to the Ingress Controller, so that all routes with labels that match the route selector are in the Ingress shard.

- Adding both a namespace selector and route selector to the Ingress Controller, so that routes with labels that match the route selector in a namespace with labels that match the namespace selector are in the Ingress shard.

With sharding, you can distribute subsets of routes over multiple Ingress Controllers. These subsets can be non–overlapping, also called *traditional* sharding, or overlapping, otherwise known as *overlapped* sharding.

8.1.1. Traditional sharding example

An Ingress Controller **finops-router** is configured with the label selector **spec.namespaceSelector.matchLabels.name** set to **finance** and **ops**:

Example YAML definition for **finops-router**

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: finops-router
  namespace: openshift-ingress-operator
spec:
  namespaceSelector:
    matchLabels:
      name:
        - finance
        - ops
```

A second Ingress Controller **dev-router** is configured with the label selector **spec.namespaceSelector.matchLabels.name** set to **dev**:

Example YAML definition for **dev-router**

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: dev-router
  namespace: openshift-ingress-operator
spec:
  namespaceSelector:
    matchLabels:
      name: dev
```

If all application routes are in separate namespaces, each labeled with **name:finance**, **name:ops**, and **name:dev** respectively, this configuration effectively distributes your routes between the two Ingress Controllers. OpenShift Container Platform routes for console, authentication, and other purposes should not be handled.

In the above scenario, sharding becomes a special case of partitioning, with no overlapping subsets. Routes are divided between router shards.

> **WARNING**
>
> The **default** Ingress Controller continues to serve all routes unless the **namespaceSelector** or **routeSelector** fields contain routes that are meant for exclusion. See this Red Hat Knowledgebase solution and the section "Sharding the default Ingress Controller" for more information on how to exclude routes from the default Ingress Controller.

8.1.2. Overlapped sharding example

In addition to **finops-router** and **dev-router** in the example above, you also have **devops-router**, which is configured with the label selector **spec.namespaceSelector.matchLabels.name** set to **dev** and **ops**:

Example YAML definition for **devops-router**

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: devops-router
  namespace: openshift-ingress-operator
spec:
  namespaceSelector:
    matchLabels:
      name:
        - dev
        - ops
```

The routes in the namespaces labeled **name:dev** and **name:ops** are now serviced by two different Ingress Controllers. With this configuration, you have overlapping subsets of routes.

With overlapping subsets of routes you can create more complex routing rules. For example, you can divert higher priority traffic to the dedicated **finops-router** while sending lower priority traffic to **devops-router**.

8.1.3. Sharding the default Ingress Controller

After creating a new Ingress shard, there might be routes that are admitted to your new Ingress shard that are also admitted by the default Ingress Controller. This is because the default Ingress Controller has no selectors and admits all routes by default.

You can restrict an Ingress Controller from servicing routes with specific labels using either namespace selectors or route selectors. The following procedure restricts the default Ingress Controller from servicing your newly sharded **finance**, **ops**, and **dev**, routes using a namespace selector. This adds further isolation to Ingress shards.

> **IMPORTANT**
>
> You must keep all of OpenShift Container Platform's administration routes on the same Ingress Controller. Therefore, avoid adding additional selectors to the default Ingress Controller that exclude these essential routes.

Prerequisites

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

- You are logged in as a project administrator.

Procedure

1. Modify the default Ingress Controller by running the following command:

   ```
   $ oc edit ingresscontroller -n openshift-ingress-operator default
   ```

2. Edit the Ingress Controller to contain a **namespaceSelector** that excludes the routes with any of the **finance**, **ops**, and **dev** labels:

   ```
   apiVersion: operator.openshift.io/v1
   kind: IngressController
   metadata:
     name: default
     namespace: openshift-ingress-operator
   spec:
     namespaceSelector:
       matchExpressions:
         - key: type
           operator: NotIn
           values:
             - finance
             - ops
             - dev
   ```

The default Ingress Controller will no longer serve the namespaces labeled **name:finance**, **name:ops**, and **name:dev**.

8.1.4. Ingress sharding and DNS

The cluster administrator is responsible for making a separate DNS entry for each router in a project. A router will not forward unknown routes to another router.

Consider the following example:

- Router A lives on host 192.168.0.5 and has routes with ***.foo.com**.

- Router B lives on host 192.168.1.9 and has routes with ***.example.com**.

Separate DNS entries must resolve ***.foo.com** to the node hosting Router A and ***.example.com** to the node hosting Router B:

- ***.foo.com A IN 192.168.0.5**

- ***.example.com A IN 192.168.1.9**

8.1.5. Configuring Ingress Controller sharding by using route labels

Ingress Controller sharding by using route labels means that the Ingress Controller serves any route in any namespace that is selected by the route selector.

Figure 8.1. Ingress sharding using route labels

Ingress Controller sharding is useful when balancing incoming traffic load among a set of Ingress Controllers and when isolating traffic to a specific Ingress Controller. For example, company A goes to one Ingress Controller and company B to another.

Procedure

1. Edit the **router-internal.yaml** file:

```
# cat router-internal.yaml
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: sharded
  namespace: openshift-ingress-operator
spec:
  domain: <apps-sharded.basedomain.example.net>  1
  nodePlacement:
    nodeSelector:
      matchLabels:
        node-role.kubernetes.io/worker: ""
  routeSelector:
    matchLabels:
      type: sharded
```

1 Specify a domain to be used by the Ingress Controller. This domain must be different from the default Ingress Controller domain.

2. Apply the Ingress Controller **router-internal.yaml** file:

   ```
   # oc apply -f router-internal.yaml
   ```

   The Ingress Controller selects routes in any namespace that have the label **type: sharded**.

3. Create a new route using the domain configured in the **router-internal.yaml**:

   ```
   $ oc expose svc <service-name> --hostname <route-name>.apps-
   sharded.basedomain.example.net
   ```

8.1.6. Configuring Ingress Controller sharding by using namespace labels

Ingress Controller sharding by using namespace labels means that the Ingress Controller serves any route in any namespace that is selected by the namespace selector.

Figure 8.2. Ingress sharding using namespace labels

Ingress Controller sharding is useful when balancing incoming traffic load among a set of Ingress Controllers and when isolating traffic to a specific Ingress Controller. For example, company A goes to one Ingress Controller and company B to another.

Procedure

1. Edit the **router-internal.yaml** file:

   ```
   # cat router-internal.yaml
   ```

   Example output

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: sharded
  namespace: openshift-ingress-operator
spec:
  domain: <apps-sharded.basedomain.example.net> 1
  nodePlacement:
    nodeSelector:
      matchLabels:
        node-role.kubernetes.io/worker: ""
  namespaceSelector:
    matchLabels:
      type: sharded
```

1   Specify a domain to be used by the Ingress Controller. This domain must be different from the default Ingress Controller domain.

2. Apply the Ingress Controller **router-internal.yaml** file:

   ```
   # oc apply -f router-internal.yaml
   ```

   The Ingress Controller selects routes in any namespace that is selected by the namespace selector that have the label **type: sharded**.

3. Create a new route using the domain configured in the **router-internal.yaml**:

   ```
   $ oc expose svc <service-name> --hostname <route-name>.apps-sharded.basedomain.example.net
   ```

8.2. CREATING A ROUTE FOR INGRESS CONTROLLER SHARDING

A route allows you to host your application at a URL. In this case, the hostname is not set and the route uses a subdomain instead. When you specify a subdomain, you automatically use the domain of the Ingress Controller that exposes the route. For situations where a route is exposed by multiple Ingress Controllers, the route is hosted at multiple URLs.

The following procedure describes how to create a route for Ingress Controller sharding, using the **hello-openshift** application as an example.

Ingress Controller sharding is useful when balancing incoming traffic load among a set of Ingress Controllers and when isolating traffic to a specific Ingress Controller. For example, company A goes to one Ingress Controller and company B to another.

Prerequisites

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

- You are logged in as a project administrator.

- You have a web application that exposes a port and an HTTP or TLS endpoint listening for traffic on the port.

- You have configured the Ingress Controller for sharding.

Procedure

1. Create a project called **hello-openshift** by running the following command:

   ```
   $ oc new-project hello-openshift
   ```

2. Create a pod in the project by running the following command:

   ```
   $ oc create -f https://raw.githubusercontent.com/openshift/origin/master/examples/hello-openshift/hello-pod.json
   ```

3. Create a service called **hello-openshift** by running the following command:

   ```
   $ oc expose pod/hello-openshift
   ```

4. Create a route definition called **hello-openshift-route.yaml**:

   YAML definition of the created route for sharding:

   ```
   apiVersion: route.openshift.io/v1
   kind: Route
   metadata:
     labels:
       type: sharded 1
     name: hello-openshift-edge
     namespace: hello-openshift
   spec:
     subdomain: hello-openshift 2
     tls:
       termination: edge
     to:
       kind: Service
       name: hello-openshift
   ```

   1 Both the label key and its corresponding label value must match the ones specified in the Ingress Controller. In this example, the Ingress Controller has the label key and value **type: sharded**.

   2 The route will be exposed using the value of the **subdomain** field. When you specify the **subdomain** field, you must leave the hostname unset. If you specify both the **host** and **subdomain** fields, then the route will use the value of the **host** field, and ignore the **subdomain** field.

5. Use **hello-openshift-route.yaml** to create a route to the **hello-openshift** application by running the following command:

   ```
   $ oc -n hello-openshift create -f hello-openshift-route.yaml
   ```

Verification

- Get the status of the route with the following command:

```
$ oc -n hello-openshift get routes/hello-openshift-edge -o yaml
```

The resulting **Route** resource should look similar to the following:

Example output

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  labels:
    type: sharded
  name: hello-openshift-edge
  namespace: hello-openshift
spec:
  subdomain: hello-openshift
  tls:
    termination: edge
  to:
    kind: Service
    name: hello-openshift
status:
  ingress:
  - host: hello-openshift.<apps-sharded.basedomain.example.net> 1
    routerCanonicalHostname: router-sharded.<apps-sharded.basedomain.example.net> 2
    routerName: sharded 3
```

1 The hostname the Ingress Controller, or router, uses to expose the route. The value of the **host** field is automatically determined by the Ingress Controller, and uses its domain. In this example, the domain of the Ingress Controller is **<apps-sharded.basedomain.example.net>**.

2 The hostname of the Ingress Controller.

3 The name of the Ingress Controller. In this example, the Ingress Controller has the name **sharded**.

Additional Resources

- Baseline Ingress Controller (router) performance

CHAPTER 9. INGRESS NODE FIREWALL OPERATOR IN OPENSHIFT CONTAINER PLATFORM

The Ingress Node Firewall Operator allows administrators to manage firewall configurations at the node level.

9.1. INGRESS NODE FIREWALL OPERATOR

The Ingress Node Firewall Operator provides ingress firewall rules at a node level by deploying the daemon set to nodes you specify and manage in the firewall configurations. To deploy the daemon set, you create an **IngressNodeFirewallConfig** custom resource (CR). The Operator applies the **IngressNodeFirewallConfig** CR to create ingress node firewall daemon set **daemon**, which run on all nodes that match the **nodeSelector**.

You configure **rules** of the **IngressNodeFirewall** CR and apply them to clusters using the **nodeSelector** and setting values to "true".
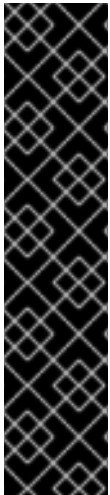
> **IMPORTANT**
>
> The Ingress Node Firewall Operator supports only stateless firewall rules.
>
> The maximum transmission units (MTU) parameter is 4Kb (kilobytes) in OpenShift Container Platform 4.13.
>
> Network interface controllers (NICs) that do not support native XDP drivers will run at a lower performance.
>
> Ingress Node Firewall Operator is not supported on Amazon Web Services (AWS) with the default OpenShift installation or on Red Hat OpenShift Service on AWS (ROSA). For more information on Red Hat OpenShift Service on AWS support and ingress, see Ingress Operator in Red Hat OpenShift Service on AWS .

9.2. INSTALLING THE INGRESS NODE FIREWALL OPERATOR

As a cluster administrator, you can install the Ingress Node Firewall Operator by using the OpenShift Container Platform CLI or the web console.

9.2.1. Installing the Ingress Node Firewall Operator using the CLI

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

Prerequisites

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

- You have an account with administrator privileges.

Procedure

1. To create the **openshift-ingress-node-firewall** namespace, enter the following command:

   ```
   $ cat << EOF| oc create -f -
   apiVersion: v1
   ```

```
kind: Namespace
metadata:
  labels:
    pod-security.kubernetes.io/enforce: privileged
    pod-security.kubernetes.io/enforce-version: v1.24
  name: openshift-ingress-node-firewall
EOF
```

2. To create an **OperatorGroup** CR, enter the following command:

```
$ cat << EOF| oc create -f -
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: ingress-node-firewall-operators
  namespace: openshift-ingress-node-firewall
EOF
```

3. Subscribe to the Ingress Node Firewall Operator.

   a. To create a **Subscription** CR for the Ingress Node Firewall Operator, enter the following command:

   ```
   $ cat << EOF| oc create -f -
   apiVersion: operators.coreos.com/v1alpha1
   kind: Subscription
   metadata:
     name: ingress-node-firewall-sub
     namespace: openshift-ingress-node-firewall
   spec:
     name: ingress-node-firewall
     channel: stable
     source: redhat-operators
     sourceNamespace: openshift-marketplace
   EOF
   ```

4. To verify that the Operator is installed, enter the following command:

```
$ oc get ip -n openshift-ingress-node-firewall
```

Example output

```
NAME         CSV                               APPROVAL   APPROVED
install-5cvnz   ingress-node-firewall.4.13.0-202211122336   Automatic   true
```

5. To verify the version of the Operator, enter the following command:

```
$ oc get csv -n openshift-ingress-node-firewall
```

Example output

```
NAME                           DISPLAY              VERSION         REPLACES
PHASE
```

> ingress-node-firewall.4.13.0-202211122336   Ingress Node Firewall Operator   4.13.0-
> 202211122336   ingress-node-firewall.4.13.0-202211102047   Succeeded

9.2.2. Installing the Ingress Node Firewall Operator using the web console

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

Prerequisites

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

- You have an account with administrator privileges.

Procedure

1. Install the Ingress Node Firewall Operator:

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

   b. Select **Ingress Node Firewall Operator** from the list of available Operators, and then click **Install**.

   c. On the **Install Operator** page, under **Installed Namespace**, select **Operator recommended Namespace**.

   d. Click **Install**.

2. Verify that the Ingress Node Firewall Operator is installed successfully:

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

   b. Ensure that **Ingress Node Firewall Operator** is listed in the **openshift-ingress-node-firewall** project with a **Status** of **InstallSucceeded**.

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

   If the Operator does not have a **Status** of **InstallSucceeded**, troubleshoot using the following steps:

   - Inspect the **Operator Subscriptions** and **Install Plans** tabs for any failures or errors under **Status**.

   - Navigate to the **Workloads → Pods** page and check the logs for pods in the **openshift-ingress-node-firewall** project.

   - Check the namespace of the YAML file. If the annotation is missing, you can add the annotation **workload.openshift.io/allowed=management** to the Operator namespace with the following command:

     ```
     $ oc annotate ns/openshift-ingress-node-firewall
     workload.openshift.io/allowed=management
     ```

NOTE

For single-node OpenShift clusters, the **openshift-ingress-node-firewall** namespace requires the **workload.openshift.io/allowed=management** annotation.

9.3. DEPLOYING INGRESS NODE FIREWALL OPERATOR

Prerequisite

- The Ingress Node Firewall Operator is installed.

Procedure

To deploy the Ingress Node Firewall Operator, create a **IngressNodeFirewallConfig** custom resource that will deploy the Operator's daemon set. You can deploy one or multiple **IngressNodeFirewall** CRDs to nodes by applying firewall rules.

1. Create the **IngressNodeFirewallConfig** inside the **openshift-ingress-node-firewall** namespace named **ingressnodefirewallconfig**.

2. Run the following command to deploy Ingress Node Firewall Operator rules:

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

9.3.1. Ingress Node Firewall configuration object

The fields for the Ingress Node Firewall configuration object are described in the following table:

Table 9.1. Ingress Node Firewall Configuration object

Field	Type	Description
metadata.name	**string**	The name of the CR object. The name of the firewall rules object must be **ingressnodefirewallconfig**.
metadata.name space	**string**	Namespace for the Ingress Firewall Operator CR object. The **IngressNodeFirewallConfig** CR must be created inside the **openshift-ingress-node-firewall** namespace.

Field	Type	Description

spec.nodeSelector	**string**	A node selection constraint used to target nodes through specified node labels. For example:
		
		``
		spec:
		  nodeSelector:
		    node-role.kubernetes.io/worker: ""
		``
		
		NOTE
		
		One label used in **nodeSelector** must match a label on the nodes in order for the daemon set to start. For example, if the node labels **node-role.kubernetes.io/worker** and **node-type.kubernetes.io/vm** are applied to a node, then at least one label must be set using **nodeSelector** for the daemon set to start.

NOTE

The Operator consumes the CR and creates an ingress node firewall daemon set on all the nodes that match the **nodeSelector**.

Ingress Node Firewall Operator example configuration

A complete Ingress Node Firewall Configuration is specified in the following example:

Example Ingress Node Firewall Configuration object

```
apiVersion: ingressnodefirewall.openshift.io/v1alpha1
kind: IngressNodeFirewallConfig
metadata:
  name: ingressnodefirewallconfig
  namespace: openshift-ingress-node-firewall
spec:
  nodeSelector:
    node-role.kubernetes.io/worker: ""
```

NOTE

The Operator consumes the CR and creates an ingress node firewall daemon set on all the nodes that match the **nodeSelector**.

9.3.2. Ingress Node Firewall rules object

The fields for the Ingress Node Firewall rules object are described in the following table:

Table 9.2. Ingress Node Firewall rules object

Field	Type	Description
metadata.name	**string**	The name of the CR object.
interfaces	**array**	The fields for this object specify the interfaces to apply the firewall rules to. For example, **- en0** and **- en1**.
nodeSelector	**array**	You can use **nodeSelector** to select the nodes to apply the firewall rules to. Set the value of your named **nodeselector** labels to **true** to apply the rule.
ingress	**object**	**ingress** allows you to configure the rules that allow outside access to the services on your cluster.

Ingress object configuration
The values for the **ingress** object are defined in the following table:

Table 9.3. **ingress** object

Field	Type	Description
sourceCIDRs	**array**	Allows you to set the CIDR block. You can configure multiple CIDRs from different address families. **NOTE** Different CIDRs allow you to use the same order rule. In the case that there are multiple **IngressNodeFirewall** objects for the same nodes and interfaces with overlapping CIDRs, the **order** field will specify which rule is applied first. Rules are applied in ascending order.

Field	Type	Description
rules	**array**	Ingress firewall **rules.order** objects are ordered starting at **1** for each **source.CIDR** with up to 100 rules per CIDR. Lower order rules are executed first. **rules.protocolConfig.protocol** supports the following protocols: TCP, UDP, SCTP, ICMP and ICMPv6. ICMP and ICMPv6 rules can match against ICMP and ICMPv6 types or codes. TCP, UDP, and SCTP rules can match against a single destination port or a range of ports using **<start : end-1>** format. Set **rules.action** to **allow** to apply the rule or **deny** to disallow the rule. **NOTE** Ingress firewall rules are verified using a verification webhook that blocks any invalid configuration. The verification webhook prevents you from blocking any critical cluster services such as the API server or SSH.

Ingress Node Firewall rules object example

A complete Ingress Node Firewall configuration is specified in the following example:

Example Ingress Node Firewall configuration

```
apiVersion: ingressnodefirewall.openshift.io/v1alpha1
kind: IngressNodeFirewall
metadata:
  name: ingressnodefirewall
spec:
  interfaces:
  - eth0
  nodeSelector:
    matchLabels:
      <ingress_firewall_label_name>: <label_value> 1
  ingress:
  - sourceCIDRs:
      - 172.16.0.0/12
    rules:
    - order: 10
      protocolConfig:
        protocol: ICMP
        icmp:
          icmpType: 8 #ICMP Echo request
      action: Deny
    - order: 20
      protocolConfig:
        protocol: TCP
```

```
      tcp:
        ports: "8000-9000"
      action: Deny
  - sourceCIDRs:
      - fc00:f853:ccd:e793::0/64
    rules:
    - order: 10
      protocolConfig:
        protocol: ICMPv6
        icmpv6:
          icmpType: 128 #ICMPV6 Echo request
      action: Deny
```

[1] A <label_name> and a <label_value> must exist on the node and must match the **nodeselector** label and value applied to the nodes you want the **ingressfirewallconfig** CR to run on. The <label_value> can be **true** or **false**. By using **nodeSelector** labels, you can target separate groups of nodes to apply different rules to using the **ingressfirewallconfig** CR.

Zero trust Ingress Node Firewall rules object example

Zero trust Ingress Node Firewall rules can provide additional security to multi-interface clusters. For example, you can use zero trust Ingress Node Firewall rules to drop all traffic on a specific interface except for SSH.

A complete configuration of a zero trust Ingress Node Firewall rule set is specified in the following example:

> **IMPORTANT**
>
> Users need to add all ports their application will use to their allowlist in the following case to ensure proper functionality.

Example zero trust Ingress Node Firewall rules

```
apiVersion: ingressnodefirewall.openshift.io/v1alpha1
kind: IngressNodeFirewall
metadata:
 name: ingressnodefirewall-zero-trust
spec:
 interfaces:
 - eth1 1
 nodeSelector:
   matchLabels:
     <ingress_firewall_label_name>: <label_value> 2
 ingress:
 - sourceCIDRs:
     - 0.0.0.0/0 3
   rules:
   - order: 10
     protocolConfig:
       protocol: TCP
       tcp:
         ports: 22
```

```
    action: Allow
  - order: 20
    action: Deny  4
```

1   Network-interface cluster

2   The <label_name> and <label_value> needs to match the **nodeSelector** label and value applied to the specific nodes with which you wish to apply the **ingressfirewallconfig** CR.

3   **0.0.0.0/0** set to match any CIDR

4   **action** set to **Deny**

9.4. VIEWING INGRESS NODE FIREWALL OPERATOR RULES

Procedure

1. Run the following command to view all current rules :

   ```
   $ oc get ingressnodefirewall
   ```

2. Choose one of the returned **<resource>** names and run the following command to view the rules or configs:

   ```
   $ oc get <resource> <name> -o yaml
   ```

9.5. TROUBLESHOOTING THE INGRESS NODE FIREWALL OPERATOR

- Run the following command to list installed Ingress Node Firewall custom resource definitions (CRD):

   ```
   $ oc get crds | grep ingressnodefirewall
   ```

   Example output

   ```
   NAME              READY  UP-TO-DATE  AVAILABLE  AGE
   ingressnodefirewallconfigs.ingressnodefirewall.openshift.io       2022-08-25T10:03:01Z
   ingressnodefirewallnodestates.ingressnodefirewall.openshift.io    2022-08-25T10:03:00Z
   ingressnodefirewalls.ingressnodefirewall.openshift.io             2022-08-25T10:03:00Z
   ```

- Run the following command to view the state of the Ingress Node Firewall Operator:

   ```
   $ oc get pods -n openshift-ingress-node-firewall
   ```

   Example output

   ```
   NAME                              READY  STATUS     RESTARTS  AGE
   ingress-node-firewall-controller-manager  2/2    Running    0      5d21h
   ingress-node-firewall-daemon-pqx56        3/3    Running    0      5d21h
   ```

The following fields provide information about the status of the Operator: **READY**, **STATUS**, **AGE**, and **RESTARTS**. The **STATUS** field is **Running** when the Ingress Node Firewall Operator is deploying a daemon set to the assigned nodes.

- Run the following command to collect all ingress firewall node pods' logs:

```
$ oc adm must-gather – gather_ingress_node_firewall
```

The logs are available in the sos node's report containing eBPF **bpftool** outputs at **/sos_commands/ebpf**. These reports include lookup tables used or updated as the ingress firewall XDP handles packet processing, updates statistics, and emits events.

CHAPTER 10. CONFIGURING AN INGRESS CONTROLLER FOR MANUAL DNS MANAGEMENT

As a cluster administrator, when you create an Ingress Controller, the Operator manages the DNS records automatically. This has some limitations when the required DNS zone is different from the cluster DNS zone or when the DNS zone is hosted outside the cloud provider.

As a cluster administrator, you can configure an Ingress Controller to stop automatic DNS management and start manual DNS management. Set **dnsManagementPolicy** to specify when it should be automatically or manually managed.

When you change an Ingress Controller from **Managed** to **Unmanaged** DNS management policy, the Operator does not clean up the previous wildcard DNS record provisioned on the cloud. When you change an Ingress Controller from **Unmanaged** to **Managed** DNS management policy, the Operator attempts to create the DNS record on the cloud provider if it does not exist or updates the DNS record if it already exists.

> **IMPORTANT**
>
> When you set **dnsManagementPolicy** to **unmanaged**, you have to manually manage the lifecycle of the wildcard DNS record on the cloud provider.

10.1. MANAGED DNS MANAGEMENT POLICY

The **Managed** DNS management policy for Ingress Controllers ensures that the lifecycle of the wildcard DNS record on the cloud provider is automatically managed by the Operator.

10.2. UNMANAGED DNS MANAGEMENT POLICY

The **Unmanaged** DNS management policy for Ingress Controllers ensures that the lifecycle of the wildcard DNS record on the cloud provider is not automatically managed, instead it becomes the responsibility of the cluster administrator.

> **NOTE**
>
> On the AWS cloud platform, if the domain on the Ingress Controller does not match with **dnsConfig.Spec.BaseDomain** then the DNS management policy is automatically set to **Unmanaged**.

10.3. CREATING A CUSTOM INGRESS CONTROLLER WITH THE UNMANAGED DNS MANAGEMENT POLICY

As a cluster administrator, you can create a new custom Ingress Controller with the **Unmanaged** DNS management policy.

Prerequisites

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

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

Procedure

1. Create a custom resource (CR) file named **sample-ingress.yaml** containing the following:

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  namespace: openshift-ingress-operator
  name: <name> ❶
spec:
  domain: <domain> ❷
  endpointPublishingStrategy:
    type: LoadBalancerService
    loadBalancer:
      scope: External ❸
      dnsManagementPolicy: Unmanaged ❹
```

❶ Specify the **<name>** with a name for the **IngressController** object.

❷ Specify the **domain** based on the DNS record that was created as a prerequisite.

❸ Specify the **scope** as **External** to expose the load balancer externally.

❹ **dnsManagementPolicy** indicates if the Ingress Controller is managing the lifecycle of the wildcard DNS record associated with the load balancer. The valid values are **Managed** and **Unmanaged**. The default value is **Managed**.

2. Save the file to apply the changes.

```
oc apply -f <name>.yaml ❶
```

10.4. MODIFYING AN EXISTING INGRESS CONTROLLER

As a cluster administrator, you can modify an existing Ingress Controller to manually manage the DNS record lifecycle.

Prerequisites

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

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

Procedure

1. Modify the chosen **IngressController** to set **dnsManagementPolicy**:

```
SCOPE=$(oc -n openshift-ingress-operator get ingresscontroller <name> -o=jsonpath="
{.status.endpointPublishingStrategy.loadBalancer.scope}")

oc -n openshift-ingress-operator patch ingresscontrollers/<name> --type=merge --
patch='{"spec":{"endpointPublishingStrategy":{"type":"LoadBalancerService","loadBalancer":
{"dnsManagementPolicy":"Unmanaged", "scope":"${SCOPE}"}}}}'
```

2. Optional: You can delete the associated DNS record in the cloud provider.

10.5. ADDITIONAL RESOURCES

- Ingress Controller configuration parameters

CHAPTER 11. CONFIGURING THE INGRESS CONTROLLER ENDPOINT PUBLISHING STRATEGY

11.1. INGRESS CONTROLLER ENDPOINT PUBLISHING STRATEGY

NodePortService endpoint publishing strategy

The **NodePortService** endpoint publishing strategy publishes the Ingress Controller using a Kubernetes NodePort service.

In this configuration, the Ingress Controller deployment uses container networking. A **NodePortService** is created to publish the deployment. The specific node ports are dynamically allocated by OpenShift Container Platform; however, to support static port allocations, your changes to the node port field of the managed **NodePortService** are preserved.

Figure 11.1. Diagram of NodePortService
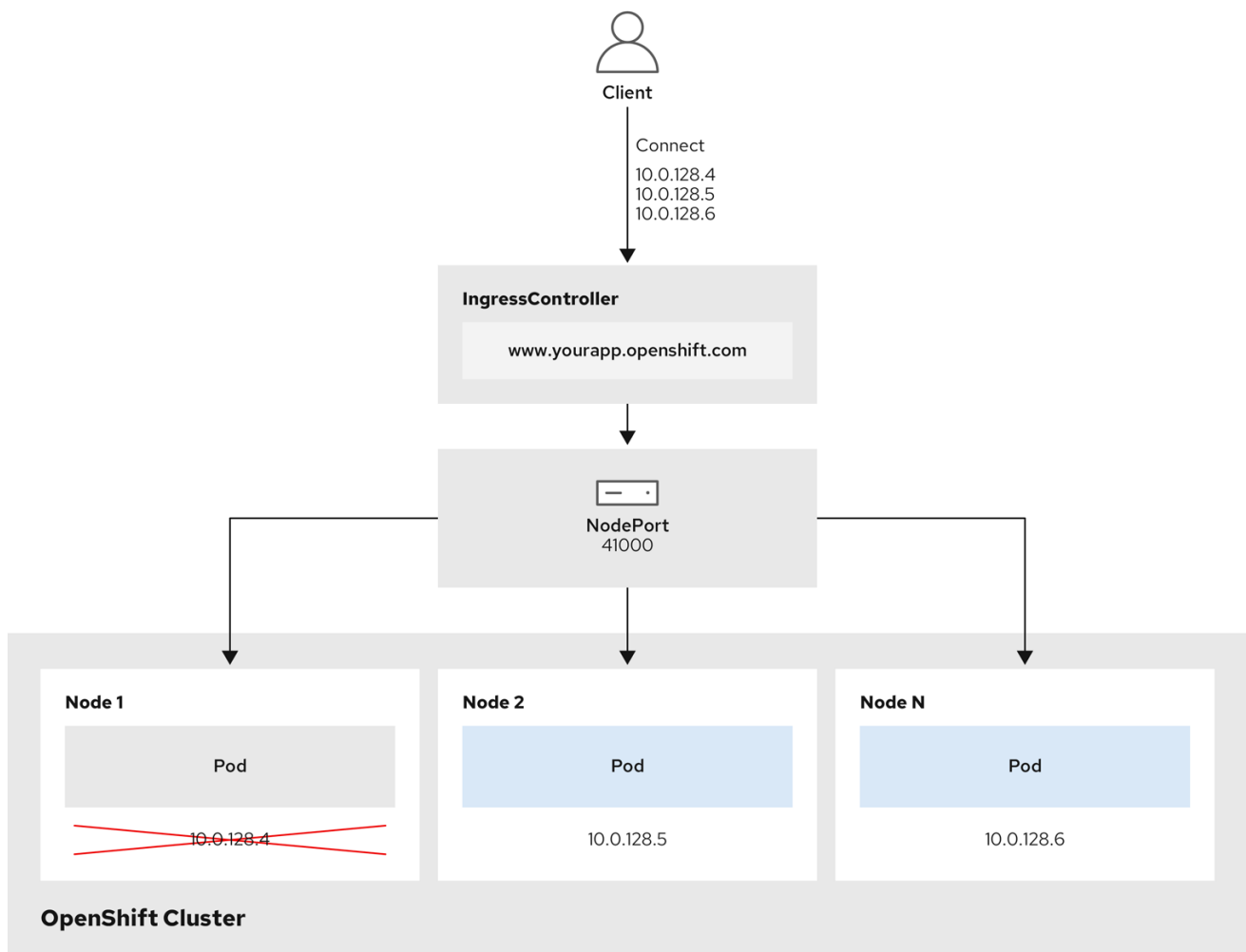
The preceding graphic shows the following concepts pertaining to OpenShift Container Platform Ingress NodePort endpoint publishing strategy:

- All the available nodes in the cluster have their own, externally accessible IP addresses. The service running in the cluster is bound to the unique NodePort for all the nodes.

- When the client connects to a node that is down, for example, by connecting the **10.0.128.4** IP

address in the graphic, the node port directly connects the client to an available node that is running the service. In this scenario, no load balancing is required. As the image shows, the **10.0.128.4** address is down and another IP address must be used instead.

> **NOTE**
>
> The Ingress Operator ignores any updates to **.spec.ports[].nodePort** fields of the service.
>
> By default, ports are allocated automatically and you can access the port allocations for integrations. However, sometimes static port allocations are necessary to integrate with existing infrastructure which may not be easily reconfigured in response to dynamic ports. To achieve integrations with static node ports, you can update the managed service resource directly.

For more information, see the Kubernetes Services documentation on **NodePort**.

HostNetwork endpoint publishing strategy

The **HostNetwork** endpoint publishing strategy publishes the Ingress Controller on node ports where the Ingress Controller is deployed.

An Ingress Controller with the **HostNetwork** endpoint publishing strategy can have only one pod replica per node. If you want *n* replicas, you must use at least *n* nodes where those replicas can be scheduled. Because each pod replica requests ports **80** and **443** on the node host where it is scheduled, a replica cannot be scheduled to a node if another pod on the same node is using those ports.

11.1.1. Configuring the Ingress Controller endpoint publishing scope to Internal

When a cluster administrator installs a new cluster without specifying that the cluster is private, the default Ingress Controller is created with a **scope** set to **External**. Cluster administrators can change an **External** scoped Ingress Controller to **Internal**.

Prerequisites

- You installed the **oc** CLI.

Procedure

- To change an **External** scoped Ingress Controller to **Internal**, enter the following command:

  ```
  $ oc -n openshift-ingress-operator patch ingresscontrollers/default --type=merge --
  patch='{"spec":{"endpointPublishingStrategy":{"type":"LoadBalancerService","loadBalancer":
  {"scope":"Internal"}}}}'
  ```

- To check the status of the Ingress Controller, enter the following command:

  ```
  $ oc -n openshift-ingress-operator get ingresscontrollers/default -o yaml
  ```

  - The **Progressing** status condition indicates whether you must take further action. For example, the status condition can indicate that you need to delete the service by entering the following command:

    ```
    $ oc -n openshift-ingress delete services/router-default
    ```

If you delete the service, the Ingress Operator recreates it as **Internal**.

11.1.2. Configuring the Ingress Controller endpoint publishing scope to External

When a cluster administrator installs a new cluster without specifying that the cluster is private, the default Ingress Controller is created with a **scope** set to **External**.

The Ingress Controller's scope can be configured to be **Internal** during installation or after, and cluster administrators can change an **Internal** Ingress Controller to **External**.

> IMPORTANT
>
> On some platforms, it is necessary to delete and recreate the service.
>
> Changing the scope can cause disruption to Ingress traffic, potentially for several minutes. This applies to platforms where it is necessary to delete and recreate the service, because the procedure can cause OpenShift Container Platform to deprovision the existing service load balancer, provision a new one, and update DNS.

Prerequisites

- You installed the **oc** CLI.

Procedure

- To change an **Internal** scoped Ingress Controller to **External**, enter the following command:

  ```
  $ oc -n openshift-ingress-operator patch ingresscontrollers/private --type=merge --patch='{"spec":{"endpointPublishingStrategy":{"type":"LoadBalancerService","loadBalancer":{"scope":"External"}}}}'
  ```

- To check the status of the Ingress Controller, enter the following command:

  ```
  $ oc -n openshift-ingress-operator get ingresscontrollers/default -o yaml
  ```

  - The **Progressing** status condition indicates whether you must take further action. For example, the status condition can indicate that you need to delete the service by entering the following command:

    ```
    $ oc -n openshift-ingress delete services/router-default
    ```

    If you delete the service, the Ingress Operator recreates it as **External**.

11.2. ADDITIONAL RESOURCES

- For more information, see Ingress Controller configuration parameters.

CHAPTER 12. VERIFYING CONNECTIVITY TO AN ENDPOINT

The Cluster Network Operator (CNO) runs a controller, the connectivity check controller, that performs a connection health check between resources within your cluster. By reviewing the results of the health checks, you can diagnose connection problems or eliminate network connectivity as the cause of an issue that you are investigating.

12.1. CONNECTION HEALTH CHECKS PERFORMED

To verify that cluster resources are reachable, a TCP connection is made to each of the following cluster API services:

- Kubernetes API server service

- Kubernetes API server endpoints

- OpenShift API server service

- OpenShift API server endpoints

- Load balancers

To verify that services and service endpoints are reachable on every node in the cluster, a TCP connection is made to each of the following targets:

- Health check target service

- Health check target endpoints

12.2. IMPLEMENTATION OF CONNECTION HEALTH CHECKS

The connectivity check controller orchestrates connection verification checks in your cluster. The results for the connection tests are stored in **PodNetworkConnectivity** objects in the **openshift-network-diagnostics** namespace. Connection tests are performed every minute in parallel.

The Cluster Network Operator (CNO) deploys several resources to the cluster to send and receive connectivity health checks:

Health check source

This program deploys in a single pod replica set managed by a **Deployment** object. The program consumes **PodNetworkConnectivity** objects and connects to the **spec.targetEndpoint** specified in each object.

Health check target

A pod deployed as part of a daemon set on every node in the cluster. The pod listens for inbound health checks. The presence of this pod on every node allows for the testing of connectivity to each node.

12.3. PODNETWORKCONNECTIVITYCHECK OBJECT FIELDS

The **PodNetworkConnectivityCheck** object fields are described in the following tables.

Table 12.1. PodNetworkConnectivityCheck object fields

Field	Type	Description
metadata.name	**string**	The name of the object in the following format: **<source>-to-<target>**. The destination described by **<target>** includes one of following strings: • **load-balancer-api-external** • **load-balancer-api-internal** • **kubernetes-apiserver-endpoint** • **kubernetes-apiserver-service-cluster** • **network-check-target** • **openshift-apiserver-endpoint** • **openshift-apiserver-service-cluster**
metadata.namespace	**string**	The namespace that the object is associated with. This value is always **openshift-network-diagnostics**.
spec.sourcePod	**string**	The name of the pod where the connection check originates, such as **network-check-source-596b4c6566-rgh92**.
spec.targetEndpoint	**string**	The target of the connection check, such as **api.devcluster.example.com:6443**.
spec.tlsClientCert	**object**	Configuration for the TLS certificate to use.
spec.tlsClientCert.name	**string**	The name of the TLS certificate used, if any. The default value is an empty string.
status	**object**	An object representing the condition of the connection test and logs of recent connection successes and failures.
status.conditions	**array**	The latest status of the connection check and any previous statuses.
status.failures	**array**	Connection test logs from unsuccessful attempts.
status.outages	**array**	Connect test logs covering the time periods of any outages.
status.successes	**array**	Connection test logs from successful attempts.

The following table describes the fields for objects in the **status.conditions** array:

Table 12.2. status.conditions

Field	Type	Description
lastTransitionTime	**string**	The time that the condition of the connection transitioned from one status to another.
message	**string**	The details about last transition in a human readable format.
reason	**string**	The last status of the transition in a machine readable format.
status	**string**	The status of the condition.
type	**string**	The type of the condition.

The following table describes the fields for objects in the **status.conditions** array:

Table 12.3. status.outages

Field	Type	Description
end	**string**	The timestamp from when the connection failure is resolved.
endLogs	**array**	Connection log entries, including the log entry related to the successful end of the outage.
message	**string**	A summary of outage details in a human readable format.
start	**string**	The timestamp from when the connection failure is first detected.
startLogs	**array**	Connection log entries, including the original failure.

Connection log fields

The fields for a connection log entry are described in the following table. The object is used in the following fields:

- **status.failures[]**

- **status.successes[]**

- **status.outages[].startLogs[]**

- **status.outages[].endLogs[]**

Table 12.4. Connection log object

Field	Type	Description
latency	**string**	Records the duration of the action.
message	**string**	Provides the status in a human readable format.
reason	**string**	Provides the reason for status in a machine readable format. The value is one of **TCPConnect**, **TCPConnectError**, **DNSResolve**, **DNSError**.
success	**boolean**	Indicates if the log entry is a success or failure.
time	**string**	The start time of connection check.

12.4. VERIFYING NETWORK CONNECTIVITY FOR AN ENDPOINT

As a cluster administrator, you can verify the connectivity of an endpoint, such as an API server, load balancer, service, or pod.

Prerequisites

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

- Access to the cluster as a user with the **cluster-admin** role.

Procedure

1. To list the current **PodNetworkConnectivityCheck** objects, enter the following command:

   ```
   $ oc get podnetworkconnectivitycheck -n openshift-network-diagnostics
   ```

   Example output

   ```
   NAME                                                                                                AGE
   network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-kubernetes-apiserver-
   endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0   75m
   network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-kubernetes-apiserver-
   endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-1   73m
   network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-kubernetes-apiserver-
   endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-2   75m
   network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-kubernetes-apiserver-
   service-cluster                     75m
   network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-kubernetes-default-
   service-cluster                     75m
   network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-load-balancer-api-
   external                        75m
   network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-load-balancer-api-
   internal                        75m
   network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-network-check-target-ci-
   ln-x5sv9rb-f76d1-4rzrp-master-0       75m
   network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-network-check-target-ci-
   ```

ln-x5sv9rb-f76d1-4rzrp-master-1   75m
network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-network-check-target-ci-
ln-x5sv9rb-f76d1-4rzrp-master-2   75m
network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-network-check-target-ci-
ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh   74m
network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-network-check-target-ci-
ln-x5sv9rb-f76d1-4rzrp-worker-c-n8mbf   74m
network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-network-check-target-ci-
ln-x5sv9rb-f76d1-4rzrp-worker-d-4hnrz   74m
network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-network-check-target-
service-cluster   75m
network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-openshift-apiserver-
endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0   75m
network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-openshift-apiserver-
endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-1   75m
network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-openshift-apiserver-
endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-2   74m
network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-openshift-apiserver-
service-cluster   75m

2. View the connection test logs:

   a. From the output of the previous command, identify the endpoint that you want to review the connectivity logs for.

   b. To view the object, enter the following command:

   ```
   $ oc get podnetworkconnectivitycheck <name> \
     -n openshift-network-diagnostics -o yaml
   ```

   where **<name>** specifies the name of the **PodNetworkConnectivityCheck** object.

   Example output

   ```
   apiVersion: controlplane.operator.openshift.io/v1alpha1
   kind: PodNetworkConnectivityCheck
   metadata:
     name: network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-kubernetes-
   apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0
     namespace: openshift-network-diagnostics
     ...
   spec:
     sourcePod: network-check-source-7c88f6d9f-hmg2f
     targetEndpoint: 10.0.0.4:6443
     tlsClientCert:
       name: ""
   status:
     conditions:
     - lastTransitionTime: "2021-01-13T20:11:34Z"
       message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0: tcp
         connection to 10.0.0.4:6443 succeeded'
       reason: TCPConnectSuccess
       status: "True"
       type: Reachable
     failures:
     - latency: 2.241775ms
   ```

      message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0: failed
        to establish a TCP connection to 10.0.0.4:6443: dial tcp 10.0.0.4:6443: connect:
        connection refused'
      reason: TCPConnectError
      success: false
      time: "2021-01-13T20:10:34Z"
    - latency: 2.582129ms
      message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0: failed
        to establish a TCP connection to 10.0.0.4:6443: dial tcp 10.0.0.4:6443: connect:
        connection refused'
      reason: TCPConnectError
      success: false
      time: "2021-01-13T20:09:34Z"
    - latency: 3.483578ms
      message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0: failed
        to establish a TCP connection to 10.0.0.4:6443: dial tcp 10.0.0.4:6443: connect:
        connection refused'
      reason: TCPConnectError
      success: false
      time: "2021-01-13T20:08:34Z"
  outages:
  - end: "2021-01-13T20:11:34Z"
    endLogs:
    - latency: 2.032018ms
      message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0:
        tcp connection to 10.0.0.4:6443 succeeded'
      reason: TCPConnect
      success: true
      time: "2021-01-13T20:11:34Z"
    - latency: 2.241775ms
      message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0:
        failed to establish a TCP connection to 10.0.0.4:6443: dial tcp 10.0.0.4:6443:
        connect: connection refused'
      reason: TCPConnectError
      success: false
      time: "2021-01-13T20:10:34Z"
    - latency: 2.582129ms
      message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0:
        failed to establish a TCP connection to 10.0.0.4:6443: dial tcp 10.0.0.4:6443:
        connect: connection refused'
      reason: TCPConnectError
      success: false
      time: "2021-01-13T20:09:34Z"
    - latency: 3.483578ms
      message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0:
        failed to establish a TCP connection to 10.0.0.4:6443: dial tcp 10.0.0.4:6443:
        connect: connection refused'
      reason: TCPConnectError
      success: false
      time: "2021-01-13T20:08:34Z"
    message: Connectivity restored after 2m59.999789186s
    start: "2021-01-13T20:08:34Z"
    startLogs:
    - latency: 3.483578ms
      message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0:
        failed to establish a TCP connection to 10.0.0.4:6443: dial tcp 10.0.0.4:6443:

    connect: connection refused'
    reason: TCPConnectError
    success: false
    time: "2021-01-13T20:08:34Z"
successes:
- latency: 2.845865ms
  message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0: tcp
    connection to 10.0.0.4:6443 succeeded'
  reason: TCPConnect
  success: true
  time: "2021-01-13T21:14:34Z"
- latency: 2.926345ms
  message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0: tcp
    connection to 10.0.0.4:6443 succeeded'
  reason: TCPConnect
  success: true
  time: "2021-01-13T21:13:34Z"
- latency: 2.895796ms
  message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0: tcp
    connection to 10.0.0.4:6443 succeeded'
  reason: TCPConnect
  success: true
  time: "2021-01-13T21:12:34Z"
- latency: 2.696844ms
  message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0: tcp
    connection to 10.0.0.4:6443 succeeded'
  reason: TCPConnect
  success: true
  time: "2021-01-13T21:11:34Z"
- latency: 1.502064ms
  message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0: tcp
    connection to 10.0.0.4:6443 succeeded'
  reason: TCPConnect
  success: true
  time: "2021-01-13T21:10:34Z"
- latency: 1.388857ms
  message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0: tcp
    connection to 10.0.0.4:6443 succeeded'
  reason: TCPConnect
  success: true
  time: "2021-01-13T21:09:34Z"
- latency: 1.906383ms
  message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0: tcp
    connection to 10.0.0.4:6443 succeeded'
  reason: TCPConnect
  success: true
  time: "2021-01-13T21:08:34Z"
- latency: 2.089073ms
  message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0: tcp
    connection to 10.0.0.4:6443 succeeded'
  reason: TCPConnect
  success: true
  time: "2021-01-13T21:07:34Z"
- latency: 2.156994ms
  message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0: tcp
    connection to 10.0.0.4:6443 succeeded'

```
      reason: TCPConnect
      success: true
      time: "2021-01-13T21:06:34Z"
    - latency: 1.777043ms
      message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0: tcp
        connection to 10.0.0.4:6443 succeeded'
      reason: TCPConnect
      success: true
      time: "2021-01-13T21:05:34Z"
```

CHAPTER 13. CHANGING THE MTU FOR THE CLUSTER NETWORK

As a cluster administrator, you can change the MTU for the cluster network after cluster installation. This change is disruptive as cluster nodes must be rebooted to finalize the MTU change. You can change the MTU only for clusters using the OVN-Kubernetes or OpenShift SDN network plugins.

13.1. ABOUT THE CLUSTER MTU

During installation the maximum transmission unit (MTU) for the cluster network is detected automatically based on the MTU of the primary network interface of nodes in the cluster. You do not normally need to override the detected MTU.

You might want to change the MTU of the cluster network for several reasons:

- The MTU detected during cluster installation is not correct for your infrastructure

- Your cluster infrastructure now requires a different MTU, such as from the addition of nodes that need a different MTU for optimal performance

You can change the cluster MTU for only the OVN-Kubernetes and OpenShift SDN cluster network plugins.

13.1.1. Service interruption considerations

When you initiate an MTU change on your cluster the following effects might impact service availability:

- At least two rolling reboots are required to complete the migration to a new MTU. During this time, some nodes are not available as they restart.

- Specific applications deployed to the cluster with shorter timeout intervals than the absolute TCP timeout interval might experience disruption during the MTU change.

13.1.2. MTU value selection

When planning your MTU migration there are two related but distinct MTU values to consider.

- **Hardware MTU**: This MTU value is set based on the specifics of your network infrastructure.

- **Cluster network MTU**: This MTU value is always less than your hardware MTU to account for the cluster network overlay overhead. The specific overhead is determined by your network plugin:

  - **OVN-Kubernetes**: **100** bytes

  - **OpenShift SDN**: **50** bytes

If your cluster requires different MTU values for different nodes, you must subtract the overhead value for your network plugin from the lowest MTU value that is used by any node in your cluster. For example, if some nodes in your cluster have an MTU of **9001**, and some have an MTU of **1500**, you must set this value to **1400**.
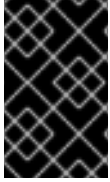
IMPORTANT

To avoid selecting an MTU value that is not acceptable by a node, verify the maximum MTU value (**maxmtu**) that is accepted by the network interface by using the **ip -d link** command.

13.1.3. How the migration process works

The following table summarizes the migration process by segmenting between the user-initiated steps in the process and the actions that the migration performs in response.

Table 13.1. Live migration of the cluster MTU

User-initiated steps	OpenShift Container Platform activity
Set the following values in the Cluster Network Operator configuration: ● **spec.migration.mtu.machine.to** ● **spec.migration.mtu.network.from** ● **spec.migration.mtu.network.to**	**Cluster Network Operator (CNO)**: Confirms that each field is set to a valid value. ● The **mtu.machine.to** must be set to either the new hardware MTU or to the current hardware MTU if the MTU for the hardware is not changing. This value is transient and is used as part of the migration process. Separately, if you specify a hardware MTU that is different from your existing hardware MTU value, you must manually configure the MTU to persist by other means, such as with a machine config, DHCP setting, or a Linux kernel command line. ● The **mtu.network.from** field must equal the **network.status.clusterNetworkMTU** field, which is the current MTU of the cluster network. ● The **mtu.network.to** field must be set to the target cluster network MTU and must be lower than the hardware MTU to allow for the overlay overhead of the network plugin. For OVN-Kubernetes, the overhead is **100** bytes and for OpenShift SDN the overhead is **50** bytes. If the values provided are valid, the CNO writes out a new temporary configuration with the MTU for the cluster network set to the value of the **mtu.network.to** field. **Machine Config Operator (MCO)**: Performs a rolling reboot of each node in the cluster.

User-initiated steps	OpenShift Container Platform activity
Reconfigure the MTU of the primary network interface for the nodes on the cluster. You can use a variety of methods to accomplish this, including: ● Deploying a new NetworkManager connection profile with the MTU change ● Changing the MTU through a DHCP server setting ● Changing the MTU through boot parameters	N/A
Set the **mtu** value in the CNO configuration for the network plugin and set **spec.migration** to **null**.	**Machine Config Operator (MCO)**: Performs a rolling reboot of each node in the cluster with the new MTU configuration.

13.2. CHANGING THE CLUSTER MTU

As a cluster administrator, you can change the maximum transmission unit (MTU) for your cluster. The migration is disruptive and nodes in your cluster might be temporarily unavailable as the MTU update rolls out.

The following procedure describes how to change the cluster MTU by using either machine configs, DHCP, or an ISO. If you use the DHCP or ISO approach, you must refer to configuration artifacts that you kept after installing your cluster to complete the procedure.

Prerequisites

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

- You are logged in to the cluster with a user with **cluster-admin** privileges.

- You identified the target MTU for your cluster. The correct MTU varies depending on the network plugin that your cluster uses:

    - **OVN-Kubernetes**: The cluster MTU must be set to **100** less than the lowest hardware MTU value in your cluster.

    - **OpenShift SDN**: The cluster MTU must be set to **50** less than the lowest hardware MTU value in your cluster.

Procedure

To increase or decrease the MTU for the cluster network complete the following procedure.

1. To obtain the current MTU for the cluster network, enter the following command:

   ```
   $ oc describe network.config cluster
   ```

   Example output

```
...
Status:
  Cluster Network:
    Cidr:          10.217.0.0/22
    Host Prefix:      23
  Cluster Network MTU:  1400
  Network Type:       OpenShiftSDN
  Service Network:
    10.217.4.0/23
...
```

2. Prepare your configuration for the hardware MTU:

- If your hardware MTU is specified with DHCP, update your DHCP configuration such as with the following dnsmasq configuration:

  ```
  dhcp-option-force=26,<mtu>
  ```

  where:

  <mtu>

    Specifies the hardware MTU for the DHCP server to advertise.

- If your hardware MTU is specified with a kernel command line with PXE, update that configuration accordingly.

- If your hardware MTU is specified in a NetworkManager connection configuration, complete the following steps. This approach is the default for OpenShift Container Platform if you do not explicitly specify your network configuration with DHCP, a kernel command line, or some other method. Your cluster nodes must all use the same underlying network configuration for the following procedure to work unmodified.

  i. Find the primary network interface:

    - If you are using the OpenShift SDN network plugin, enter the following command:

      ```
      $ oc debug node/<node_name> -- chroot /host ip route list match 0.0.0.0/0 | awk '{print $5 }'
      ```

      where:

      <node_name>

        Specifies the name of a node in your cluster.

    - If you are using the OVN-Kubernetes network plugin, enter the following command:

      ```
      $ oc debug node/<node_name> -- chroot /host nmcli -g connection.interface-name c show ovs-if-phys0
      ```

      where:

      <node_name>

        Specifies the name of a node in your cluster.

  ii. Create the following NetworkManager configuration in the **<interface>-mtu.conf** file:

Example NetworkManager connection configuration

```
[connection-<interface>-mtu]
match-device=interface-name:<interface>
ethernet.mtu=<mtu>
```

where:

<mtu>

Specifies the new hardware MTU value.

<interface>

Specifies the primary network interface name.

iii. Create two **MachineConfig** objects, one for the control plane nodes and another for the worker nodes in your cluster:

A. Create the following Butane config in the **control-plane-interface.bu** file:

```
variant: openshift
version: 4.13.0
metadata:
  name: 01-control-plane-interface
  labels:
    machineconfiguration.openshift.io/role: master
storage:
  files:
    - path: /etc/NetworkManager/conf.d/99-<interface>-mtu.conf  1
      contents:
        local: <interface>-mtu.conf  2
      mode: 0600
```

1 **1** Specify the NetworkManager connection name for the primary network interface.

2 Specify the local filename for the updated NetworkManager configuration file from the previous step.

B. Create the following Butane config in the **worker-interface.bu** file:

```
variant: openshift
version: 4.13.0
metadata:
  name: 01-worker-interface
  labels:
    machineconfiguration.openshift.io/role: worker
storage:
  files:
    - path: /etc/NetworkManager/conf.d/99-<interface>-mtu.conf  1
      contents:
        local: <interface>-mtu.conf  2
      mode: 0600
```

1 Specify the NetworkManager connection name for the primary network interface.

2 Specify the local filename for the updated NetworkManager configuration file from the previous step.

C. Create **MachineConfig** objects from the Butane configs by running the following command:

```
$ for manifest in control-plane-interface worker-interface; do
    butane --files-dir . $manifest.bu > $manifest.yaml
  done
```

3. To begin the MTU migration, specify the migration configuration by entering the following command. The Machine Config Operator performs a rolling reboot of the nodes in the cluster in preparation for the MTU change.

```
$ oc patch Network.operator.openshift.io cluster --type=merge --patch \
  '{"spec": { "migration": { "mtu": { "network": { "from": <overlay_from>, "to": <overlay_to> } ,
"machine": { "to" : <machine_to> } } } } }'
```

where:

<overlay_from>

Specifies the current cluster network MTU value.

<overlay_to>

Specifies the target MTU for the cluster network. This value is set relative to the value for **<machine_to>** and for OVN-Kubernetes must be **100** less and for OpenShift SDN must be **50** less.

<machine_to>

Specifies the MTU for the primary network interface on the underlying host network.

Example that increases the cluster MTU

```
$ oc patch Network.operator.openshift.io cluster --type=merge --patch \
  '{"spec": { "migration": { "mtu": { "network": { "from": 1400, "to": 9000 } , "machine": { "to" :
9100} } } } }'
```

4. As the MCO updates machines in each machine config pool, it reboots each node one by one. You must wait until all the nodes are updated. Check the machine config pool status by entering the following command:

```
$ oc get mcp
```

A successfully updated node has the following status: **UPDATED=true**, **UPDATING=false**, **DEGRADED=false**.

> **NOTE**
>
> By default, the MCO updates one machine per pool at a time, causing the total time the migration takes to increase with the size of the cluster.

5. Confirm the status of the new machine configuration on the hosts:

   a. To list the machine configuration state and the name of the applied machine configuration, enter the following command:

   ```
   $ oc describe node | egrep "hostname|machineconfig"
   ```

   Example output

   ```
   kubernetes.io/hostname=master-0
   machineconfiguration.openshift.io/currentConfig: rendered-master-
   c53e221d9d24e1c8bb6ee89dd3d8ad7b
   machineconfiguration.openshift.io/desiredConfig: rendered-master-
   c53e221d9d24e1c8bb6ee89dd3d8ad7b
   machineconfiguration.openshift.io/reason:
   machineconfiguration.openshift.io/state: Done
   ```

   Verify that the following statements are true:

   - The value of **machineconfiguration.openshift.io/state** field is **Done**.

   - The value of the **machineconfiguration.openshift.io/currentConfig** field is equal to the value of the **machineconfiguration.openshift.io/desiredConfig** field.

   b. To confirm that the machine config is correct, enter the following command:

   ```
   $ oc get machineconfig <config_name> -o yaml | grep ExecStart
   ```

   where **<config_name>** is the name of the machine config from the **machineconfiguration.openshift.io/currentConfig** field.

   The machine config must include the following update to the systemd configuration:

   ```
   ExecStart=/usr/local/bin/mtu-migration.sh
   ```

6. Update the underlying network interface MTU value:

   - If you are specifying the new MTU with a NetworkManager connection configuration, enter the following command. The MachineConfig Operator automatically performs a rolling reboot of the nodes in your cluster.

   ```
   $ for manifest in control-plane-interface worker-interface; do
       oc create -f $manifest.yaml
     done
   ```

   - If you are specifying the new MTU with a DHCP server option or a kernel command line and PXE, make the necessary changes for your infrastructure.

7. As the MCO updates machines in each machine config pool, it reboots each node one by one. You must wait until all the nodes are updated. Check the machine config pool status by entering the following command:

   ```
   $ oc get mcp
   ```

A successfully updated node has the following status: **UPDATED=true**, **UPDATING=false**, **DEGRADED=false**.
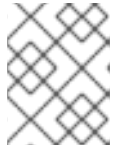
> **NOTE**
>
> By default, the MCO updates one machine per pool at a time, causing the total time the migration takes to increase with the size of the cluster.

8. Confirm the status of the new machine configuration on the hosts:

   a. To list the machine configuration state and the name of the applied machine configuration, enter the following command:

   ```
   $ oc describe node | egrep "hostname|machineconfig"
   ```

   Example output

   ```
   kubernetes.io/hostname=master-0
   machineconfiguration.openshift.io/currentConfig: rendered-master-
   c53e221d9d24e1c8bb6ee89dd3d8ad7b
   machineconfiguration.openshift.io/desiredConfig: rendered-master-
   c53e221d9d24e1c8bb6ee89dd3d8ad7b
   machineconfiguration.openshift.io/reason:
   machineconfiguration.openshift.io/state: Done
   ```

   Verify that the following statements are true:

   - The value of **machineconfiguration.openshift.io/state** field is **Done**.

   - The value of the **machineconfiguration.openshift.io/currentConfig** field is equal to the value of the **machineconfiguration.openshift.io/desiredConfig** field.

   b. To confirm that the machine config is correct, enter the following command:

   ```
   $ oc get machineconfig <config_name> -o yaml | grep path:
   ```

   where **<config_name>** is the name of the machine config from the **machineconfiguration.openshift.io/currentConfig** field.

   If the machine config is successfully deployed, the previous output contains the **/etc/NetworkManager/conf.d/99-<interface>-mtu.conf** file path and the **ExecStart=/usr/local/bin/mtu-migration.sh** line.

9. To finalize the MTU migration, enter one of the following commands:

   - If you are using the OVN-Kubernetes network plugin:

   ```
   $ oc patch Network.operator.openshift.io cluster --type=merge --patch \
     '{"spec": { "migration": null, "defaultNetwork":{ "ovnKubernetesConfig": { "mtu": <mtu>
   }}}}'
   ```

   where:

   <mtu>

Specifies the new cluster network MTU that you specified with **<overlay_to>**.

- If you are using the OpenShift SDN network plugin:

```
$ oc patch Network.operator.openshift.io cluster --type=merge --patch \
  '{"spec": { "migration": null, "defaultNetwork":{ "openshiftSDNConfig": { "mtu": <mtu> }}}}'
```

where:

<mtu>

Specifies the new cluster network MTU that you specified with **<overlay_to>**.

10. After finalizing the MTU migration, each MCP node is rebooted one by one. You must wait until all the nodes are updated. Check the machine config pool status by entering the following command:

```
$ oc get mcp
```

A successfully updated node has the following status: **UPDATED=true**, **UPDATING=false**, **DEGRADED=false**.

Verification

You can verify that a node in your cluster uses an MTU that you specified in the previous procedure.

1. To get the current MTU for the cluster network, enter the following command:

```
$ oc describe network.config cluster
```

2. Get the current MTU for the primary network interface of a node.

   a. To list the nodes in your cluster, enter the following command:

   ```
   $ oc get nodes
   ```

   b. To obtain the current MTU setting for the primary network interface on a node, enter the following command:

   ```
   $ oc debug node/<node> -- chroot /host ip address show <interface>
   ```

   where:

   <node>

   Specifies a node from the output from the previous step.

   <interface>

   Specifies the primary network interface name for the node.

   Example output

   ```
   ens3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 8051
   ```

13.3. ADDITIONAL RESOURCES

- Using advanced networking options for PXE and ISO installations

- Manually creating NetworkManager profiles in key file format

- Configuring a dynamic Ethernet connection using nmcli

- Using advanced networking options for PXE and ISO installations

- Manually creating NetworkManager profiles in key file format

- Configuring a dynamic Ethernet connection using nmcli

CHAPTER 14. CONFIGURING THE NODE PORT SERVICE RANGE

As a cluster administrator, you can expand the available node port range. If your cluster uses of a large number of node ports, you might need to increase the number of available ports.

The default port range is **30000-32767**. You can never reduce the port range, even if you first expand it beyond the default range.

14.1. PREREQUISITES

- Your cluster infrastructure must allow access to the ports that you specify within the expanded range. For example, if you expand the node port range to **30000-32900**, the inclusive port range of **32768-32900** must be allowed by your firewall or packet filtering configuration.

14.2. EXPANDING THE NODE PORT RANGE

You can expand the node port range for the cluster.

Prerequisites

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

- Log in to the cluster with a user with **cluster-admin** privileges.

Procedure

1. To expand the node port range, enter the following command. Replace **<port>** with the largest port number in the new range.

    ```
    $ oc patch network.config.openshift.io cluster --type=merge -p \
      '{
        "spec":
          { "serviceNodePortRange": "30000-<port>" }
      }'
    ```

    TIP

    You can alternatively apply the following YAML to update the node port range:

    ```
    apiVersion: config.openshift.io/v1
    kind: Network
    metadata:
      name: cluster
    spec:
      serviceNodePortRange: "30000-<port>"
    ```

    Example output

    ```
    network.config.openshift.io/cluster patched
    ```

2. To confirm that the configuration is active, enter the following command. It can take several minutes for the update to apply.

```
$ oc get configmaps -n openshift-kube-apiserver config \
  -o jsonpath="{.data['config\.yaml']}" | \
  grep -Eo '"service-node-port-range":["[[:digit:]]+-[[:digit:]]+"]'
```

Example output

```
"service-node-port-range":["30000-33000"]
```

14.3. ADDITIONAL RESOURCES

- Configuring ingress cluster traffic using a NodePort

- Network [config.openshift.io/v1]

- Service [core/v1]

CHAPTER 15. CONFIGURING THE CLUSTER NETWORK RANGE

As a cluster administrator, you can expand the cluster network range after cluster installation. You might want to expand the cluster network range if you need more IP addresses for additional nodes.

For example, if you deployed a cluster and specified **10.128.0.0/19** as the cluster network range and a host prefix of **23**, you are limited to 16 nodes. You can expand that to 510 nodes by changing the CIDR mask on a cluster to **/14**.

When expanding the cluster network address range, your cluster must use the OVN-Kubernetes network plugin. Other network plugins are not supported.

The following limitations apply when modifying the cluster network IP address range:

- The CIDR mask size specified must always be smaller than the currently configured CIDR mask size, because you can only increase IP space by adding more nodes to an installed cluster

- The host prefix cannot be modified

- Pods that are configured with an overridden default gateway must be recreated after the cluster network expands

15.1. EXPANDING THE CLUSTER NETWORK IP ADDRESS RANGE

You can expand the IP address range for the cluster network. Because this change requires rolling out a new Operator configuration across the cluster, it can take up to 30 minutes to take effect.

Prerequisites

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

- Log in to the cluster with a user with **cluster-admin** privileges.

- Ensure that the cluster uses the OVN-Kubernetes network plugin.

Procedure

1. To obtain the cluster network range and host prefix for your cluster, enter the following command:

   ```
   $ oc get network.operator.openshift.io \
     -o jsonpath="{.items[0].spec.clusterNetwork}"
   ```

   Example output

   ```
   [{"cidr":"10.217.0.0/22","hostPrefix":23}]
   ```

2. To expand the cluster network IP address range, enter the following command. Use the CIDR IP address range and host prefix returned from the output of the previous command.

   ```
   $ oc patch Network.config.openshift.io cluster --type='merge' --patch \
     '{
       "spec":{
         "clusterNetwork": [ {"cidr":"<network>/<cidr>","hostPrefix":<prefix>} ],
   ```

```
        "networkType": "OVNKubernetes"
      }
    }'
```

where:

<network>

Specifies the network part of the **cidr** field that you obtained from the previous step. You cannot change this value.

<cidr>

Specifies the network prefix length. For example, **14**. Change this value to a smaller number than the value from the output in the previous step to expand the cluster network range.

<prefix>

Specifies the current host prefix for your cluster. This value must be the same value for the **hostPrefix** field that you obtained from the previous step.

Example command

```
$ oc patch Network.config.openshift.io cluster --type='merge' --patch \
  '{
    "spec":{
      "clusterNetwork": [ {"cidr":"10.217.0.0/14","hostPrefix": 23} ],
      "networkType": "OVNKubernetes"
    }
  }'
```

Example output

```
network.config.openshift.io/cluster patched
```

3. To confirm that the configuration is active, enter the following command. It can take up to 30 minutes for this change to take effect.

```
$ oc get network.operator.openshift.io \
  -o jsonpath="{.items[0].spec.clusterNetwork}"
```

Example output

```
[{"cidr":"10.217.0.0/14","hostPrefix":23}]
```

15.2. ADDITIONAL RESOURCES

- About the OVN-Kubernetes network plugin

CHAPTER 16. CONFIGURING IP FAILOVER

This topic describes configuring IP failover for pods and services on your OpenShift Container Platform cluster.

IP failover manages a pool of Virtual IP (VIP) addresses on a set of nodes. Every VIP in the set is serviced by a node selected from the set. As long a single node is available, the VIPs are served. There is no way to explicitly distribute the VIPs over the nodes, so there can be nodes with no VIPs and other nodes with many VIPs. If there is only one node, all VIPs are on it.

> **NOTE**
>
> The VIPs must be routable from outside the cluster.

IP failover monitors a port on each VIP to determine whether the port is reachable on the node. If the port is not reachable, the VIP is not assigned to the node. If the port is set to **0**, this check is suppressed. The check script does the needed testing.

IP failover uses Keepalived to host a set of externally accessible VIP addresses on a set of hosts. Each VIP is only serviced by a single host at a time. Keepalived uses the Virtual Router Redundancy Protocol (VRRP) to determine which host, from the set of hosts, services which VIP. If a host becomes unavailable, or if the service that Keepalived is watching does not respond, the VIP is switched to another host from the set. This means a VIP is always serviced as long as a host is available.

When a node running Keepalived passes the check script, the VIP on that node can enter the **master** state based on its priority and the priority of the current master and as determined by the preemption strategy.

A cluster administrator can provide a script through the **OPENSHIFT_HA_NOTIFY_SCRIPT** variable, and this script is called whenever the state of the VIP on the node changes. Keepalived uses the **master** state when it is servicing the VIP, the **backup** state when another node is servicing the VIP, or in the **fault** state when the check script fails. The notify script is called with the new state whenever the state changes.

You can create an IP failover deployment configuration on OpenShift Container Platform. The IP failover deployment configuration specifies the set of VIP addresses, and the set of nodes on which to service them. A cluster can have multiple IP failover deployment configurations, with each managing its own set of unique VIP addresses. Each node in the IP failover configuration runs an IP failover pod, and this pod runs Keepalived.

When using VIPs to access a pod with host networking, the application pod runs on all nodes that are running the IP failover pods. This enables any of the IP failover nodes to become the master and service the VIPs when needed. If application pods are not running on all nodes with IP failover, either some IP failover nodes never service the VIPs or some application pods never receive any traffic. Use the same selector and replication count, for both IP failover and the application pods, to avoid this mismatch.

While using VIPs to access a service, any of the nodes can be in the IP failover set of nodes, since the service is reachable on all nodes, no matter where the application pod is running. Any of the IP failover nodes can become master at any time. The service can either use external IPs and a service port or it can use a **NodePort**.

When using external IPs in the service definition, the VIPs are set to the external IPs, and the IP failover monitoring port is set to the service port. When using a node port, the port is open on every node in the cluster, and the service load-balances traffic from whatever node currently services the VIP. In this case, the IP failover monitoring port is set to the **NodePort** in the service definition.

IMPORTANT

Setting up a **NodePort** is a privileged operation.

IMPORTANT

Even though a service VIP is highly available, performance can still be affected. Keepalived makes sure that each of the VIPs is serviced by some node in the configuration, and several VIPs can end up on the same node even when other nodes have none. Strategies that externally load-balance across a set of VIPs can be thwarted when IP failover puts multiple VIPs on the same node.

When you use **ingressIP**, you can set up IP failover to have the same VIP range as the **ingressIP** range. You can also disable the monitoring port. In this case, all the VIPs appear on same node in the cluster. Any user can set up a service with an **ingressIP** and have it highly available.
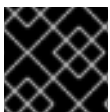
IMPORTANT

There are a maximum of 254 VIPs in the cluster.

16.1. IP FAILOVER ENVIRONMENT VARIABLES

The following table contains the variables used to configure IP failover.

Table 16.1. IP failover environment variables

Variable Name	Default	Description
OPENSHIFT_HA_MONITOR_PORT	80	The IP failover pod tries to open a TCP connection to this port on each Virtual IP (VIP). If connection is established, the service is considered to be running. If this port is set to **0**, the test always passes.
OPENSHIFT_HA_NETWORK_INTERFACE		The interface name that IP failover uses to send Virtual Router Redundancy Protocol (VRRP) traffic. The default value is **eth0**.
OPENSHIFT_HA_REPLICA_COUNT	2	The number of replicas to create. This must match **spec.replicas** value in IP failover deployment configuration.
OPENSHIFT_HA_VIRTUAL_IPS		The list of IP address ranges to replicate. This must be provided. For example, **1.2.3.4-6,1.2.3.9**.
OPENSHIFT_HA_VRRP_ID_OFFSET	0	The offset value used to set the virtual router IDs. Using different offset values allows multiple IP failover configurations to exist within the same cluster. The default offset is **0**, and the allowed range is **0** through **255**.

Variable Name	Default	Description
OPENSHIFT_HA_VIP_GROUPS		The number of groups to create for VRRP. If not set, a group is created for each virtual IP range specified with the **OPENSHIFT_HA_VIP_GROUPS** variable.
OPENSHIFT_HA_IPTABLES_CHAIN	INPUT	The name of the iptables chain, to automatically add an **iptables** rule to allow the VRRP traffic on. If the value is not set, an **iptables** rule is not added. If the chain does not exist, it is not created.
OPENSHIFT_HA_CHECK_SCRIPT		The full path name in the pod file system of a script that is periodically run to verify the application is operating.
OPENSHIFT_HA_CHECK_INTERVAL	**2**	The period, in seconds, that the check script is run.
OPENSHIFT_HA_NOTIFY_SCRIPT		The full path name in the pod file system of a script that is run whenever the state changes.
OPENSHIFT_HA_PREEMPTION	**preempt _nodelay 300**	The strategy for handling a new higher priority host. The **nopreempt** strategy does not move master from the lower priority host to the higher priority host.

16.2. CONFIGURING IP FAILOVER

As a cluster administrator, you can configure IP failover on an entire cluster, or on a subset of nodes, as defined by the label selector. You can also configure multiple IP failover deployment configurations in your cluster, where each one is independent of the others.

The IP failover deployment configuration ensures that a failover pod runs on each of the nodes matching the constraints or the label used.

This pod runs Keepalived, which can monitor an endpoint and use Virtual Router Redundancy Protocol (VRRP) to fail over the virtual IP (VIP) from one node to another if the first node cannot reach the service or endpoint.

For production use, set a **selector** that selects at least two nodes, and set **replicas** equal to the number of selected nodes.

Prerequisites

- You are logged in to the cluster with a user with **cluster-admin** privileges.

- You created a pull secret.

Procedure

Procedure

1. Create an IP failover service account:

   ```
   $ oc create sa ipfailover
   ```

2. Update security context constraints (SCC) for **hostNetwork**:

   ```
   $ oc adm policy add-scc-to-user privileged -z ipfailover
   $ oc adm policy add-scc-to-user hostnetwork -z ipfailover
   ```

3. Create a deployment YAML file to configure IP failover:

   Example deployment YAML for IP failover configuration

   ```
   apiVersion: apps/v1
   kind: Deployment
   metadata:
     name: ipfailover-keepalived ❶
     labels:
       ipfailover: hello-openshift
   spec:
     strategy:
       type: Recreate
     replicas: 2
     selector:
       matchLabels:
         ipfailover: hello-openshift
     template:
       metadata:
         labels:
           ipfailover: hello-openshift
       spec:
         serviceAccountName: ipfailover
         privileged: true
         hostNetwork: true
         nodeSelector:
           node-role.kubernetes.io/worker: ""
         containers:
         - name: openshift-ipfailover
           image: quay.io/openshift/origin-keepalived-ipfailover
           ports:
           - containerPort: 63000
             hostPort: 63000
           imagePullPolicy: IfNotPresent
           securityContext:
             privileged: true
           volumeMounts:
           - name: lib-modules
             mountPath: /lib/modules
             readOnly: true
           - name: host-slash
             mountPath: /host
             readOnly: true
             mountPropagation: HostToContainer
           - name: etc-sysconfig
   ```

```
      mountPath: /etc/sysconfig
      readOnly: true
    - name: config-volume
      mountPath: /etc/keepalive
    env:
    - name: OPENSHIFT_HA_CONFIG_NAME
      value: "ipfailover"
    - name: OPENSHIFT_HA_VIRTUAL_IPS 2
      value: "1.1.1.1-2"
    - name: OPENSHIFT_HA_VIP_GROUPS 3
      value: "10"
    - name: OPENSHIFT_HA_NETWORK_INTERFACE 4
      value: "ens3" #The host interface to assign the VIPs
    - name: OPENSHIFT_HA_MONITOR_PORT 5
      value: "30060"
    - name: OPENSHIFT_HA_VRRP_ID_OFFSET 6
      value: "0"
    - name: OPENSHIFT_HA_REPLICA_COUNT 7
      value: "2" #Must match the number of replicas in the deployment
    - name: OPENSHIFT_HA_USE_UNICAST
      value: "false"
    #- name: OPENSHIFT_HA_UNICAST_PEERS
      #value: "10.0.148.40,10.0.160.234,10.0.199.110"
    - name: OPENSHIFT_HA_IPTABLES_CHAIN 8
      value: "INPUT"
    #- name: OPENSHIFT_HA_NOTIFY_SCRIPT 9
    #  value: /etc/keepalive/mynotifyscript.sh
    - name: OPENSHIFT_HA_CHECK_SCRIPT 10
      value: "/etc/keepalive/mycheckscript.sh"
    - name: OPENSHIFT_HA_PREEMPTION 11
      value: "preempt_delay 300"
    - name: OPENSHIFT_HA_CHECK_INTERVAL 12
      value: "2"
    livenessProbe:
      initialDelaySeconds: 10
      exec:
        command:
        - pgrep
        - keepalived
volumes:
- name: lib-modules
  hostPath:
    path: /lib/modules
- name: host-slash
  hostPath:
    path: /
- name: etc-sysconfig
  hostPath:
    path: /etc/sysconfig
# config-volume contains the check script
# created with `oc create configmap keepalived-checkscript --from-file=mycheckscript.sh`
- configMap:
    defaultMode: 0755
    name: keepalived-checkscript
```

```
    name: config-volume
  imagePullSecrets:
    - name: openshift-pull-secret 13
```

1 The name of the IP failover deployment.

2 The list of IP address ranges to replicate. This must be provided. For example, **1.2.3.4-6,1.2.3.9**.

3 The number of groups to create for VRRP. If not set, a group is created for each virtual IP range specified with the **OPENSHIFT_HA_VIP_GROUPS** variable.

4 The interface name that IP failover uses to send VRRP traffic. By default, **eth0** is used.

5 The IP failover pod tries to open a TCP connection to this port on each VIP. If connection is established, the service is considered to be running. If this port is set to **0**, the test always passes. The default value is **80**.

6 The offset value used to set the virtual router IDs. Using different offset values allows multiple IP failover configurations to exist within the same cluster. The default offset is **0**, and the allowed range is **0** through **255**.

7 The number of replicas to create. This must match **spec.replicas** value in IP failover deployment configuration. The default value is **2**.

8 The name of the **iptables** chain to automatically add an **iptables** rule to allow the VRRP traffic on. If the value is not set, an **iptables** rule is not added. If the chain does not exist, it is not created, and Keepalived operates in unicast mode. The default is **INPUT**.

9 The full path name in the pod file system of a script that is run whenever the state changes.

10 The full path name in the pod file system of a script that is periodically run to verify the application is operating.

11 The strategy for handling a new higher priority host. The default value is **preempt_delay 300**, which causes a Keepalived instance to take over a VIP after 5 minutes if a lower-priority master is holding the VIP.

12 The period, in seconds, that the check script is run. The default value is **2**.

13 Create the pull secret before creating the deployment, otherwise you will get an error when creating the deployment.

16.3. ABOUT VIRTUAL IP ADDRESSES

Keepalived manages a set of virtual IP addresses (VIP). The administrator must make sure that all of these addresses:

- Are accessible on the configured hosts from outside the cluster.

- Are not used for any other purpose within the cluster.

Keepalived on each node determines whether the needed service is running. If it is, VIPs are supported and Keepalived participates in the negotiation to determine which node serves the VIP. For a node to participate, the service must be listening on the watch port on a VIP or the check must be disabled.

> **NOTE**
>
> Each VIP in the set may end up being served by a different node.

16.4. CONFIGURING CHECK AND NOTIFY SCRIPTS

Keepalived monitors the health of the application by periodically running an optional user supplied check script. For example, the script can test a web server by issuing a request and verifying the response.

When a check script is not provided, a simple default script is run that tests the TCP connection. This default test is suppressed when the monitor port is **0**.

Each IP failover pod manages a Keepalived daemon that manages one or more virtual IPs (VIP) on the node where the pod is running. The Keepalived daemon keeps the state of each VIP for that node. A particular VIP on a particular node may be in **master**, **backup**, or **fault** state.

When the check script for that VIP on the node that is in **master** state fails, the VIP on that node enters the **fault** state, which triggers a renegotiation. During renegotiation, all VIPs on a node that are not in the **fault** state participate in deciding which node takes over the VIP. Ultimately, the VIP enters the **master** state on some node, and the VIP stays in the **backup** state on the other nodes.

When a node with a VIP in **backup** state fails, the VIP on that node enters the **fault** state. When the check script passes again for a VIP on a node in the **fault** state, the VIP on that node exits the **fault** state and negotiates to enter the **master** state. The VIP on that node may then enter either the **master** or the **backup** state.

As cluster administrator, you can provide an optional notify script, which is called whenever the state changes. Keepalived passes the following three parameters to the script:

- **$1** - **group** or **instance**

- **$2** - Name of the **group** or **instance**

- **$3** - The new state: **master**, **backup**, or **fault**

The check and notify scripts run in the IP failover pod and use the pod file system, not the host file system. However, the IP failover pod makes the host file system available under the **/hosts** mount path. When configuring a check or notify script, you must provide the full path to the script. The recommended approach for providing the scripts is to use a config map.

The full path names of the check and notify scripts are added to the Keepalived configuration file, _**/etc/keepalived/keepalived.conf**, which is loaded every time Keepalived starts. The scripts can be added to the pod with a config map as follows.

Prerequisites

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

- You are logged in to the cluster with a user with **cluster-admin** privileges.

Procedure

1. Create the desired script and create a config map to hold it. The script has no input arguments and must return **0** for **OK** and **1** for **fail**.
   The check script, *mycheckscript.sh*:

   ```
   #!/bin/bash
       # Whatever tests are needed
       # E.g., send request and verify response
   exit 0
   ```

2. Create the config map:

   ```
   $ oc create configmap mycustomcheck --from-file=mycheckscript.sh
   ```

3. Add the script to the pod. The **defaultMode** for the mounted config map files must able to run by using **oc** commands or by editing the deployment configuration. A value of **0755**, **493** decimal, is typical:

   ```
   $ oc set env deploy/ipfailover-keepalived \
       OPENSHIFT_HA_CHECK_SCRIPT=/etc/keepalive/mycheckscript.sh
   ```

   ```
   $ oc set volume deploy/ipfailover-keepalived --add --overwrite \
       --name=config-volume \
       --mount-path=/etc/keepalive \
       --source='{"configMap": { "name": "mycustomcheck", "defaultMode": 493}}'
   ```

   > **NOTE**
   >
   > The **oc set env** command is whitespace sensitive. There must be no whitespace on either side of the **=** sign.

TIP

You can alternatively edit the **ipfailover-keepalived** deployment configuration:

```
$ oc edit deploy ipfailover-keepalived
```

```
    spec:
      containers:
      - env:
        - name: OPENSHIFT_HA_CHECK_SCRIPT ❶
          value: /etc/keepalive/mycheckscript.sh
...
        volumeMounts: ❷
        - mountPath: /etc/keepalive
          name: config-volume
      dnsPolicy: ClusterFirst
...
        volumes: ❸
        - configMap:
            defaultMode: 0755 ❹
            name: customrouter
          name: config-volume
...
```

❶ In the **spec.container.env** field, add the **OPENSHIFT_HA_CHECK_SCRIPT** environment variable to point to the mounted script file.

❷ Add the **spec.container.volumeMounts** field to create the mount point.

❸ Add a new **spec.volumes** field to mention the config map.

❹ This sets run permission on the files. When read back, it is displayed in decimal, **493**.

Save the changes and exit the editor. This restarts **ipfailover-keepalived**.

16.5. CONFIGURING VRRP PREEMPTION

When a Virtual IP (VIP) on a node leaves the **fault** state by passing the check script, the VIP on the node enters the **backup** state if it has lower priority than the VIP on the node that is currently in the **master** state. However, if the VIP on the node that is leaving **fault** state has a higher priority, the preemption strategy determines its role in the cluster.

The **nopreempt** strategy does not move **master** from the lower priority VIP on the host to the higher priority VIP on the host. With **preempt_delay 300**, the default, Keepalived waits the specified 300 seconds and moves **master** to the higher priority VIP on the host.

Prerequisites

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

Procedure

- To specify preemption enter **oc edit deploy ipfailover-keepalived** to edit the router deployment configuration:

```
$ oc edit deploy ipfailover-keepalived
```

```
...
  spec:
   containers:
   - env:
     - name: OPENSHIFT_HA_PREEMPTION ❶
       value: preempt_delay 300
...
```

❶ Set the **OPENSHIFT_HA_PREEMPTION** value:

- ○ **preempt_delay 300**: Keepalived waits the specified 300 seconds and moves **master** to the higher priority VIP on the host. This is the default value.

- ○ **nopreempt**: does not move **master** from the lower priority VIP on the host to the higher priority VIP on the host.

16.6. ABOUT VRRP ID OFFSET

Each IP failover pod managed by the IP failover deployment configuration, **1** pod per node or replica, runs a Keepalived daemon. As more IP failover deployment configurations are configured, more pods are created and more daemons join into the common Virtual Router Redundancy Protocol (VRRP) negotiation. This negotiation is done by all the Keepalived daemons and it determines which nodes service which virtual IPs (VIP).

Internally, Keepalived assigns a unique **vrrp-id** to each VIP. The negotiation uses this set of **vrrp-ids**, when a decision is made, the VIP corresponding to the winning **vrrp-id** is serviced on the winning node.

Therefore, for every VIP defined in the IP failover deployment configuration, the IP failover pod must assign a corresponding **vrrp-id**. This is done by starting at **OPENSHIFT_HA_VRRP_ID_OFFSET** and sequentially assigning the **vrrp-ids** to the list of VIPs. The **vrrp-ids** can have values in the range **1..255**.

When there are multiple IP failover deployment configurations, you must specify **OPENSHIFT_HA_VRRP_ID_OFFSET** so that there is room to increase the number of VIPs in the deployment configuration and none of the **vrrp-id** ranges overlap.

16.7. CONFIGURING IP FAILOVER FOR MORE THAN 254 ADDRESSES

IP failover management is limited to 254 groups of Virtual IP (VIP) addresses. By default OpenShift Container Platform assigns one IP address to each group. You can use the **OPENSHIFT_HA_VIP_GROUPS** variable to change this so multiple IP addresses are in each group and define the number of VIP groups available for each Virtual Router Redundancy Protocol (VRRP) instance when configuring IP failover.

Grouping VIPs creates a wider range of allocation of VIPs per VRRP in the case of VRRP failover events, and is useful when all hosts in the cluster have access to a service locally. For example, when a service is being exposed with an **ExternalIP**.

> **NOTE**
>
> As a rule for failover, do not limit services, such as the router, to one specific host. Instead, services should be replicated to each host so that in the case of IP failover, the services do not have to be recreated on the new host.
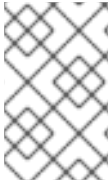
> **NOTE**
>
> If you are using OpenShift Container Platform health checks, the nature of IP failover and groups means that all instances in the group are not checked. For that reason, the Kubernetes health checks must be used to ensure that services are live.

Prerequisites

- You are logged in to the cluster with a user with **cluster-admin** privileges.

Procedure

- To change the number of IP addresses assigned to each group, change the value for the **OPENSHIFT_HA_VIP_GROUPS** variable, for example:

  Example Deployment YAML for IP failover configuration

  ```
  ...
    spec:
      env:
      - name: OPENSHIFT_HA_VIP_GROUPS  1
        value: "3"
  ...
  ```

  1 If **OPENSHIFT_HA_VIP_GROUPS** is set to **3** in an environment with seven VIPs, it creates three groups, assigning three VIPs to the first group, and two VIPs to the two remaining groups.

> **NOTE**
>
> If the number of groups set by **OPENSHIFT_HA_VIP_GROUPS** is fewer than the number of IP addresses set to fail over, the group contains more than one IP address, and all of the addresses move as a single unit.

16.8. HIGH AVAILABILITY FOR INGRESSIP

In non-cloud clusters, IP failover and **ingressIP** to a service can be combined. The result is high availability services for users that create services using **ingressIP**.

The approach is to specify an **ingressIPNetworkCIDR** range and then use the same range in creating the IP failover configuration.

Because IP failover can support up to a maximum of 255 VIPs for the entire cluster, the **ingressIPNetworkCIDR** must be **/24** or smaller.

16.9. REMOVING IP FAILOVER

When IP failover is initially configured, the worker nodes in the cluster are modified with an **iptables** rule that explicitly allows multicast packets on **224.0.0.18** for Keepalived. Because of the change to the nodes, removing IP failover requires running a job to remove the **iptables** rule and removing the virtual IP addresses used by Keepalived.

Procedure

1. Optional: Identify and delete any check and notify scripts that are stored as config maps:

   a. Identify whether any pods for IP failover use a config map as a volume:

   ```
   $ oc get pod -l ipfailover \
     -o jsonpath="\
   {range .items[?(@.spec.volumes[*].configMap)]}
   {'Namespace: '}{.metadata.namespace}
   {'Pod:       '}{.metadata.name}
   {'Volumes that use config maps:'}
   {range .spec.volumes[?(@.configMap)]}  {'volume:    '}{.name}
     {'configMap: '}{.configMap.name}{'\n'}{end}
   {end}"
   ```

   Example output

   ```
   Namespace: default
   Pod:       keepalived-worker-59df45db9c-2x9mn
   Volumes that use config maps:
     volume:   config-volume
     configMap: mycustomcheck
   ```

   b. If the preceding step provided the names of config maps that are used as volumes, delete the config maps:

   ```
   $ oc delete configmap <configmap_name>
   ```

2. Identify an existing deployment for IP failover:

   ```
   $ oc get deployment -l ipfailover
   ```

   Example output

   ```
   NAMESPACE  NAME        READY  UP-TO-DATE  AVAILABLE  AGE
   default    ipfailover  2/2    2           2          105d
   ```

3. Delete the deployment:

   ```
   $ oc delete deployment <ipfailover_deployment_name>
   ```

4. Remove the **ipfailover** service account:

   ```
   $ oc delete sa ipfailover
   ```

5. Run a job that removes the IP tables rule that was added when IP failover was initially configured:

   a. Create a file such as **remove-ipfailover-job.yaml** with contents that are similar to the following example:

   ```
   apiVersion: batch/v1
   ```

```
kind: Job
metadata:
  generateName: remove-ipfailover-
  labels:
    app: remove-ipfailover
spec:
  template:
    metadata:
      name: remove-ipfailover
    spec:
      containers:
      - name: remove-ipfailover
        image: quay.io/openshift/origin-keepalived-ipfailover:4.13
        command: ["/var/lib/ipfailover/keepalived/remove-failover.sh"]
      nodeSelector:
        kubernetes.io/hostname: <host_name>  <.>
      restartPolicy: Never
```

<.> Run the job for each node in your cluster that was configured for IP failover and replace the hostname each time.

b. Run the job:

```
$ oc create -f remove-ipfailover-job.yaml
```

Example output

```
job.batch/remove-ipfailover-2h8dm created
```

Verification

- Confirm that the job removed the initial configuration for IP failover.

```
$ oc logs job/remove-ipfailover-2h8dm
```

Example output

```
remove-failover.sh: OpenShift IP Failover service terminating.
  - Removing ip_vs module ...
  - Cleaning up ...
  - Releasing VIPs  (interface eth0) ...
```

CHAPTER 17. CONFIGURING INTERFACE-LEVEL NETWORK SYSCTLS

In Linux, sysctl allows an administrator to modify kernel parameters at runtime. You can modify interface-level network sysctls using the tuning Container Network Interface (CNI) meta plugin. The tuning CNI meta plugin operates in a chain with a main CNI plugin as illustrated.
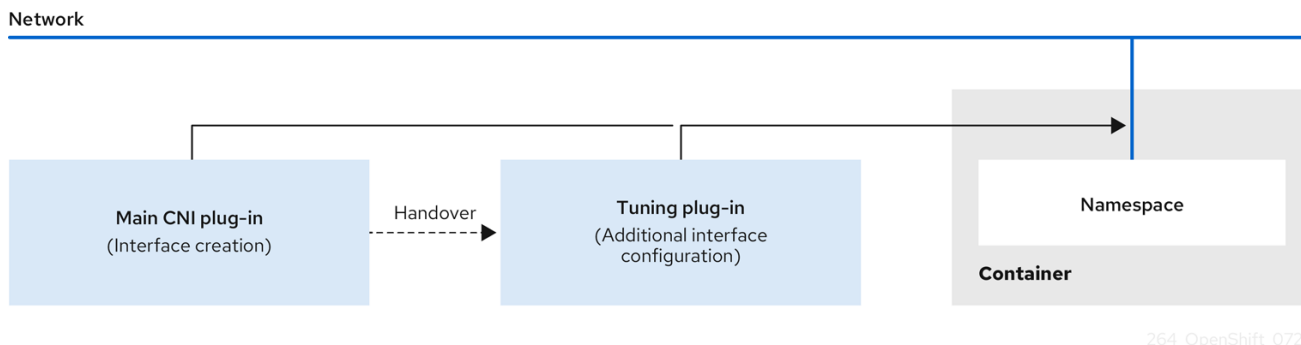
The main CNI plugin assigns the interface and passes this to the tuning CNI meta plugin at runtime. You can change some sysctls and several interface attributes (promiscuous mode, all-multicast mode, MTU, and MAC address) in the network namespace by using the tuning CNI meta plugin. In the tuning CNI meta plugin configuration, the interface name is represented by the **IFNAME** token, and is replaced with the actual name of the interface at runtime.

> **NOTE**
>
> In OpenShift Container Platform, the tuning CNI meta plugin only supports changing interface-level network sysctls.

17.1. CONFIGURING THE TUNING CNI

The following procedure configures the tuning CNI to change the interface-level network **net.ipv4.conf.IFNAME.accept_redirects** sysctl. This example enables accepting and sending ICMP-redirected packets.

Procedure

1. Create a network attachment definition, such as **tuning-example.yaml**, with the following content:

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: <name> 1
  namespace: default 2
spec:
  config: '{
    "cniVersion": "0.4.0", 3
    "name": "<name>", 4
    "plugins": [{
      "type": "<main_CNI_plugin>" 5
      },
```

```
        {
          "type": "tuning", ❻
          "sysctl": {
              "net.ipv4.conf.IFNAME.accept_redirects": "1" ❼
          }
        }
      ]
  }
```

❶ Specifies the name for the additional network attachment to create. The name must be unique within the specified namespace.

❷ Specifies the namespace that the object is associated with.

❸ Specifies the CNI specification version.

❹ Specifies the name for the configuration. It is recommended to match the configuration name to the name value of the network attachment definition.

❺ Specifies the name of the main CNI plugin to configure.

❻ Specifies the name of the CNI meta plugin.

❼ Specifies the sysctl to set.

An example yaml file is shown here:

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: tuningnad
  namespace: default
spec:
  config: '{
    "cniVersion": "0.4.0",
    "name": "tuningnad",
    "plugins": [{
      "type": "bridge"
      },
      {
      "type": "tuning",
      "sysctl": {
          "net.ipv4.conf.IFNAME.accept_redirects": "1"
      }
    }
  ]
}'
```

2. Apply the yaml by running the following command:

```
$ oc apply -f tuning-example.yaml
```

Example output

```
networkattachmentdefinition.k8.cni.cncf.io/tuningnad created
```

3. Create a pod such as **examplepod.yaml** with the network attachment definition similar to the following:

```
apiVersion: v1
kind: Pod
metadata:
  name: tunepod
  namespace: default
  annotations:
    k8s.v1.cni.cncf.io/networks: tuningnad ❶
spec:
  containers:
  - name: podexample
    image: centos
    command: ["/bin/bash", "-c", "sleep INF"]
    securityContext:
      runAsUser: 2000 ❷
      runAsGroup: 3000 ❸
      allowPrivilegeEscalation: false ❹
      capabilities: ❺
        drop: ["ALL"]
  securityContext:
    runAsNonRoot: true ❻
    seccompProfile: ❼
      type: RuntimeDefault
```

❶ Specify the name of the configured **NetworkAttachmentDefinition**.

❷ **runAsUser** controls which user ID the container is run with.

❸ **runAsGroup** controls which primary group ID the containers is run with.

❹ **allowPrivilegeEscalation** determines if a pod can request to allow privilege escalation. If unspecified, it defaults to true. This boolean directly controls whether the **no_new_privs** flag gets set on the container process.

❺ **capabilities** permit privileged actions without giving full root access. This policy ensures all capabilities are dropped from the pod.

❻ **runAsNonRoot: true** requires that the container will run with a user with any UID other than 0.

❼ **RuntimeDefault** enables the default seccomp profile for a pod or container workload.

4. Apply the yaml by running the following command:

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

5. Verify that the pod is created by running the following command:

```
$ oc get pod
```

Example output

```
NAME      READY  STATUS    RESTARTS  AGE
tunepod   1/1    Running   0         47s
```

6. Log in to the pod by running the following command:

```
$ oc rsh tunepod
```

7. Verify the values of the configured sysctl flags. For example, find the value
net.ipv4.conf.net1.accept_redirects by running the following command:

```
sh-4.4# sysctl net.ipv4.conf.net1.accept_redirects
```

Expected output

```
net.ipv4.conf.net1.accept_redirects = 1
```

17.2. ADDITIONAL RESOURCES

- [Using sysctls in containers](#)

CHAPTER 18. USING THE STREAM CONTROL TRANSMISSION PROTOCOL (SCTP) ON A BARE METAL CLUSTER

As a cluster administrator, you can use the Stream Control Transmission Protocol (SCTP) on a cluster.

18.1. SUPPORT FOR STREAM CONTROL TRANSMISSION PROTOCOL (SCTP) ON OPENSHIFT CONTAINER PLATFORM

As a cluster administrator, you can enable SCTP on the hosts in the cluster. On Red Hat Enterprise Linux CoreOS (RHCOS), the SCTP module is disabled by default.

SCTP is a reliable message based protocol that runs on top of an IP network.

When enabled, you can use SCTP as a protocol with pods, services, and network policy. A **Service** object must be defined with the **type** parameter set to either the **ClusterIP** or **NodePort** value.

18.1.1. Example configurations using SCTP protocol

You can configure a pod or service to use SCTP by setting the **protocol** parameter to the **SCTP** value in the pod or service object.

In the following example, a pod is configured to use SCTP:

```
apiVersion: v1
kind: Pod
metadata:
  namespace: project1
  name: example-pod
spec:
  containers:
    - name: example-pod
...
    ports:
      - containerPort: 30100
        name: sctpserver
        protocol: SCTP
```

In the following example, a service is configured to use SCTP:

```
apiVersion: v1
kind: Service
metadata:
  namespace: project1
  name: sctpserver
spec:
...
  ports:
    - name: sctpserver
      protocol: SCTP
      port: 30100
      targetPort: 30100
  type: ClusterIP
```

In the following example, a **NetworkPolicy** object is configured to apply to SCTP network traffic on port **80** from any pods with a specific label:

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-sctp-on-http
spec:
  podSelector:
    matchLabels:
      role: web
  ingress:
  - ports:
    - protocol: SCTP
      port: 80
```

18.2. ENABLING STREAM CONTROL TRANSMISSION PROTOCOL (SCTP)

As a cluster administrator, you can load and enable the blacklisted SCTP kernel module on worker nodes in your cluster.

Prerequisites

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

- Access to the cluster as a user with the **cluster-admin** role.

Procedure

1. Create a file named **load-sctp-module.yaml** that contains the following YAML definition:

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  name: load-sctp-module
  labels:
    machineconfiguration.openshift.io/role: worker
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
      files:
        - path: /etc/modprobe.d/sctp-blacklist.conf
          mode: 0644
          overwrite: true
          contents:
            source: data:,
        - path: /etc/modules-load.d/sctp-load.conf
          mode: 0644
          overwrite: true
          contents:
            source: data:,sctp
```

2. To create the **MachineConfig** object, enter the following command:

```
$ oc create -f load-sctp-module.yaml
```

3. Optional: To watch the status of the nodes while the MachineConfig Operator applies the configuration change, enter the following command. When the status of a node transitions to **Ready**, the configuration update is applied.

```
$ oc get nodes
```

18.3. VERIFYING STREAM CONTROL TRANSMISSION PROTOCOL (SCTP) IS ENABLED

You can verify that SCTP is working on a cluster by creating a pod with an application that listens for SCTP traffic, associating it with a service, and then connecting to the exposed service.

Prerequisites

- Access to the internet from the cluster to install the **nc** package.

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

- Access to the cluster as a user with the **cluster-admin** role.

Procedure

1. Create a pod starts an SCTP listener:

   a. Create a file named **sctp-server.yaml** that defines a pod with the following YAML:

   ```yaml
   apiVersion: v1
   kind: Pod
   metadata:
     name: sctpserver
     labels:
       app: sctpserver
   spec:
     containers:
       - name: sctpserver
         image: registry.access.redhat.com/ubi9/ubi
         command: ["/bin/sh", "-c"]
         args:
           ["dnf install -y nc && sleep inf"]
         ports:
           - containerPort: 30102
             name: sctpserver
             protocol: SCTP
   ```

   b. Create the pod by entering the following command:

   ```
   $ oc create -f sctp-server.yaml
   ```

2. Create a service for the SCTP listener pod.

a. Create a file named **sctp-service.yaml** that defines a service with the following YAML:

```
apiVersion: v1
kind: Service
metadata:
  name: sctpservice
  labels:
    app: sctpserver
spec:
  type: NodePort
  selector:
    app: sctpserver
  ports:
    - name: sctpserver
      protocol: SCTP
      port: 30102
      targetPort: 30102
```

b. To create the service, enter the following command:

```
$ oc create -f sctp-service.yaml
```

3. Create a pod for the SCTP client.

a. Create a file named **sctp-client.yaml** with the following YAML:

```
apiVersion: v1
kind: Pod
metadata:
  name: sctpclient
  labels:
    app: sctpclient
spec:
  containers:
    - name: sctpclient
      image: registry.access.redhat.com/ubi9/ubi
      command: ["/bin/sh", "-c"]
      args:
        ["dnf install -y nc && sleep inf"]
```

b. To create the **Pod** object, enter the following command:

```
$ oc apply -f sctp-client.yaml
```

4. Run an SCTP listener on the server.

a. To connect to the server pod, enter the following command:

```
$ oc rsh sctpserver
```

b. To start the SCTP listener, enter the following command:

```
$ nc -l 30102 --sctp
```

5. Connect to the SCTP listener on the server.

    a. Open a new terminal window or tab in your terminal program.

    b. Obtain the IP address of the **sctpservice** service. Enter the following command:

```
$ oc get services sctpservice -o go-template='{{.spec.clusterIP}}{{"\n"}}'
```

    c. To connect to the client pod, enter the following command:

```
$ oc rsh sctpclient
```

    d. To start the SCTP client, enter the following command. Replace **<cluster_IP>** with the cluster IP address of the **sctpservice** service.

```
# nc <cluster_IP> 30102 --sctp
```

CHAPTER 19. USING PTP HARDWARE

You can configure **linuxptp** services and use PTP-capable hardware in OpenShift Container Platform cluster nodes.

19.1. ABOUT PTP HARDWARE

You can use the OpenShift Container Platform console or OpenShift CLI (**oc**) to install PTP by deploying the PTP Operator. The PTP Operator creates and manages the **linuxptp** services and provides the following features:

- Discovery of the PTP-capable devices in the cluster.

- Management of the configuration of **linuxptp** services.

- Notification of PTP clock events that negatively affect the performance and reliability of your application with the PTP Operator **cloud-event-proxy** sidecar.

> **NOTE**
>
> The PTP Operator works with PTP-capable devices on clusters provisioned only on bare-metal infrastructure.

19.2. ABOUT PTP

Precision Time Protocol (PTP) is used to synchronize clocks in a network. When used in conjunction with hardware support, PTP is capable of sub-microsecond accuracy, and is more accurate than Network Time Protocol (NTP).

19.2.1. Elements of a PTP domain

PTP is used to synchronize multiple nodes connected in a network, with clocks for each node. The clocks synchronized by PTP are organized in a source-destination hierarchy. The hierarchy is created and updated automatically by the best master clock (BMC) algorithm, which runs on every clock. Destination clocks are synchronized to source clocks, and destination clocks can themselves be the source for other downstream clocks. The three primary types of PTP clocks are described below.

Grandmaster clock

The grandmaster clock provides standard time information to other clocks across the network and ensures accurate and stable synchronisation. It writes time stamps and responds to time requests from other clocks. Grandmaster clocks synchronize to a Global Navigation Satellite System (GNSS) time source. The Grandmaster clock is the authoritative source of time in the network and is responsible for providing time synchronization to all other devices.

Ordinary clock

The ordinary clock has a single port connection that can play the role of source or destination clock, depending on its position in the network. The ordinary clock can read and write time stamps.

Boundary clock

The boundary clock has ports in two or more communication paths and can be a source and a destination to other destination clocks at the same time. The boundary clock works as a destination clock upstream. The destination clock receives the timing message, adjusts for delay, and then

creates a new source time signal to pass down the network. The boundary clock produces a new timing packet that is still correctly synced with the source clock and can reduce the number of connected devices reporting directly to the source clock.

19.2.2. Advantages of PTP over NTP

One of the main advantages that PTP has over NTP is the hardware support present in various network interface controllers (NIC) and network switches. The specialized hardware allows PTP to account for delays in message transfer and improves the accuracy of time synchronization. To achieve the best possible accuracy, it is recommended that all networking components between PTP clocks are PTP hardware enabled.

Hardware-based PTP provides optimal accuracy, since the NIC can time stamp the PTP packets at the exact moment they are sent and received. Compare this to software-based PTP, which requires additional processing of the PTP packets by the operating system.
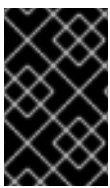
> **IMPORTANT**
>
> Before enabling PTP, ensure that NTP is disabled for the required nodes. You can disable the chrony time service (**chronyd**) using a **MachineConfig** custom resource. For more information, see Disabling chrony time service .

19.2.3. Using PTP with dual NIC hardware

OpenShift Container Platform supports single and dual NIC hardware for precision PTP timing in the cluster.

For 5G telco networks that deliver mid-band spectrum coverage, each virtual distributed unit (vDU) requires connections to 6 radio units (RUs). To make these connections, each vDU host requires 2 NICs configured as boundary clocks.

Dual NIC hardware allows you to connect each NIC to the same upstream leader clock with separate **ptp4l** instances for each NIC feeding the downstream clocks.

19.3. OVERVIEW OF LINUXPTP IN OPENSHIFT CONTAINER PLATFORM NODES

OpenShift Container Platform uses PTP and **linuxptp** for high precision system timing in bare-metal infrastructure. The **linuxptp** package includes the **ts2phc**, **pmc**, **ptp4l**, and **phc2sys** programs for system clock synchronization.

ts2phc

> **ts2phc** synchronizes the PTP hardware clock (PHC) across PTP devices with a high degree of precision. **ts2phc** is used in grandmaster clock configurations. It receives the precision timing signal a high precision clock source such as Global Navigation Satellite System (GNSS). GNSS provides an accurate and reliable source of synchronized time for use in large distributed networks. GNSS clocks typically provide time information with a precision of a few nanoseconds.
> The **ts2phc** system daemon sends timing information from the grandmaster clock to other PTP devices in the network by reading time information from the grandmaster clock and converting it to PHC format. PHC time is used by other devices in the network to synchronize their clocks with the grandmaster clock.

pmc

pmc implements a PTP management client (**pmc**) according to IEEE standard 1588.1588. **pmc** provides basic management access for the **ptp4l** system daemon. **pmc** reads from standard input and sends the output over the selected transport, printing any replies it receives.

ptp4l

ptp4l implements the PTP boundary clock and ordinary clock and runs as a system daemon. **ptp4l** does the following:

- Synchronizes the PHC to the source clock with hardware time stamping

- Synchronizes the system clock to the source clock with software time stamping

phc2sys

phc2sys synchronizes the system clock to the PHC on the network interface controller (NIC). The **phc2sys** system daemon continuously monitors the PHC for timing information. When it detects a timing error, the PHC corrects the system clock.

19.4. INSTALLING THE PTP OPERATOR USING THE CLI

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

Prerequisites

- A cluster installed on bare-metal hardware with nodes that have hardware that supports PTP.

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

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

Procedure

1. Create a namespace for the PTP Operator.

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

      ```
      apiVersion: v1
      kind: Namespace
      metadata:
       name: openshift-ptp
       annotations:
         workload.openshift.io/allowed: management
       labels:
         name: openshift-ptp
         openshift.io/cluster-monitoring: "true"
      ```

   b. Create the **Namespace** CR:

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

2. Create an Operator group for the PTP Operator.

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

      ```
      apiVersion: operators.coreos.com/v1
      kind: OperatorGroup
      ```

```
metadata:
  name: ptp-operators
  namespace: openshift-ptp
spec:
  targetNamespaces:
  - openshift-ptp
```

b. Create the **OperatorGroup** CR:

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

3. Subscribe to the PTP Operator.

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

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: ptp-operator-subscription
  namespace: openshift-ptp
spec:
  channel: "stable"
  name: ptp-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
```

b. Create the **Subscription** CR:

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

4. To verify that the Operator is installed, enter the following command:

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

Example output

```
Name                    Phase
4.13.0-202301261535     Succeeded
```

19.5. INSTALLING THE PTP OPERATOR USING THE WEB CONSOLE

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

> **NOTE**
>
> You have to create the namespace and Operator group as mentioned in the previous section.

Procedure

1. Install the PTP Operator using the OpenShift Container Platform web console:

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

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

c. On the **Install Operator** page, under **A specific namespace on the cluster** select **openshift-ptp**. Then, click **Install**.

2. Optional: Verify that the PTP Operator installed successfully:

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

b. Ensure that **PTP Operator** is listed in the **openshift-ptp** project with a **Status** of **InstallSucceeded**.

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

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

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

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

19.6. CONFIGURING PTP DEVICES

The PTP Operator adds the **NodePtpDevice.ptp.openshift.io** custom resource definition (CRD) to OpenShift Container Platform.

When installed, the PTP Operator searches your cluster for PTP-capable network devices on each node. It creates and updates a **NodePtpDevice** custom resource (CR) object for each node that provides a compatible PTP-capable network device.

19.6.1. Discovering PTP capable network devices in your cluster

- To return a complete list of PTP capable network devices in your cluster, run the following command:

```
$ oc get NodePtpDevice -n openshift-ptp -o yaml
```

Example output

```
apiVersion: v1
items:
- apiVersion: ptp.openshift.io/v1
  kind: NodePtpDevice
  metadata:
    creationTimestamp: "2022-01-27T15:16:28Z"
    generation: 1
    name: dev-worker-0 ❶
    namespace: openshift-ptp
```

```
    resourceVersion: "6538103"
    uid: d42fc9ad-bcbf-4590-b6d8-b676c642781a
  spec: {}
  status:
    devices: ❷
    - name: eno1
    - name: eno2
    - name: eno3
    - name: eno4
    - name: enp5s0f0
    - name: enp5s0f1
  ...
```

❶ The value for the **name** parameter is the same as the name of the parent node.

❷ The **devices** collection includes a list of the PTP capable devices that the PTP Operator discovers for the node.

19.6.2. Configuring linuxptp services as a grandmaster clock

You can configure the **linuxptp** services (**ptp4l**, **phc2sys**, **ts2phc**) as a grandmaster clock (T-GM) by creating a **PtpConfig** custom resource (CR) that configures the host NIC.

The **ts2phc** utility allows you to synchronize the system clock with the PTP grandmaster clock so that the node can stream precision clock signal to downstream PTP ordinary clocks and boundary clocks.

> **NOTE**
>
> Use the following example **PtpConfig** CR as the basis to configure **linuxptp** services as the grandmaster clock for your particular hardware and environment. This example CR does not configure PTP fast events. To configure PTP fast events, set appropriate values for **ptp4lOpts**, **ptp4lConf**, and **ptpClockThreshold**. **ptpClockThreshold** is used only when events are enabled. See "Configuring the PTP fast event notifications publisher" for more information.

Prerequisites

- For T-GM clocks in production environments, install an Intel E810 Westport Channel NIC in the bare-metal cluster host.

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

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

- Install the PTP Operator.

Procedure

1. Create the **PtpConfig** resource. For example:

    a. Depending on your requirements, use one of the following T-GM configurations for your deployment. Save the YAML in the **grandmaster-clock-ptp-config.yaml** file:

        Example 19.1. Example PTP grandmaster clock configuration

```
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: grandmaster-clock
  namespace: openshift-ptp
  annotations: {}
spec:
  profile:
    - name: grandmaster-clock
      # The interface name is hardware-specific
      interface: $interface
      ptp4lOpts: "-2"
      phc2sysOpts: "-a -r -r -n 24"
      ptpSchedulingPolicy: SCHED_FIFO
      ptpSchedulingPriority: 10
      ptpSettings:
        logReduce: "true"
      ptp4lConf: |
        [global]
        #
        # Default Data Set
        #
        twoStepFlag 1
        slaveOnly 0
        priority1 128
        priority2 128
        domainNumber 24
        #utc_offset 37
        clockClass 255
        clockAccuracy 0xFE
        offsetScaledLogVariance 0xFFFF
        free_running 0
        freq_est_interval 1
        dscp_event 0
        dscp_general 0
        dataset_comparison G.8275.x
        G.8275.defaultDS.localPriority 128
        #
        # Port Data Set
        #
        logAnnounceInterval -3
        logSyncInterval -4
        logMinDelayReqInterval -4
        logMinPdelayReqInterval -4
        announceReceiptTimeout 3
        syncReceiptTimeout 0
        delayAsymmetry 0
        fault_reset_interval -4
        neighborPropDelayThresh 20000000
        masterOnly 0
        G.8275.portDS.localPriority 128
        #
        # Run time options
        #
        assume_two_step 0
        logging_level 6
```

```
path_trace_enabled 0
follow_up_info 0
hybrid_e2e 0
inhibit_multicast_service 0
net_sync_monitor 0
tc_spanning_tree 0
tx_timestamp_timeout 50
unicast_listen 0
unicast_master_table 0
unicast_req_duration 3600
use_syslog 1
verbose 0
summary_interval 0
kernel_leap 1
check_fup_sync 0
clock_class_threshold 7
#
# Servo Options
#
pi_proportional_const 0.0
pi_integral_const 0.0
pi_proportional_scale 0.0
pi_proportional_exponent -0.3
pi_proportional_norm_max 0.7
pi_integral_scale 0.0
pi_integral_exponent 0.4
pi_integral_norm_max 0.3
step_threshold 2.0
first_step_threshold 0.00002
max_frequency 900000000
clock_servo pi
sanity_freq_limit 200000000
ntpshm_segment 0
#
# Transport options
#
transportSpecific 0x0
ptp_dst_mac 01:1B:19:00:00:00
p2p_dst_mac 01:80:C2:00:00:0E
udp_ttl 1
udp6_scope 0x0E
uds_address /var/run/ptp4l
#
# Default interface options
#
clock_type OC
network_transport L2
delay_mechanism E2E
time_stamping hardware
tsproc_mode filter
delay_filter moving_median
delay_filter_length 10
egressLatency 0
ingressLatency 0
boundary_clock_jbod 0
#
```

```
      # Clock description
      #
      productDescription ;;
      revisionData ;;
      manufacturerIdentity 00:00:00
      userDescription ;
      timeSource 0xA0
  recommend:
    - profile: grandmaster-clock
      priority: 4
      match:
        - nodeLabel: "node-role.kubernetes.io/$mcp"
```

> **NOTE**
>
> The example PTP grandmaster clock configuration is for test purposes
> only and is not intended for production.

Example 19.2. PTP grandmaster clock configuration for E810 NIC

```
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: grandmaster
  namespace: openshift-ptp
  annotations:
    ran.openshift.io/ztp-deploy-wave: "10"
spec:
  profile:
  - name: "grandmaster"
    ptp4lOpts: "-2 --summary_interval -4"
    phc2sysOpts: -r -u 0 -m -O -37 -N 8 -R 16 -s $iface_master -n 24
    ptpSchedulingPolicy: SCHED_FIFO
    ptpSchedulingPriority: 10
    ptpSettings:
      logReduce: "true"
    plugins:
      e810:
        enableDefaultConfig: true
    ts2phcOpts: " "
    ts2phcConf: |
      [nmea]
      ts2phc.master 1
      [global]
      use_syslog  0
      verbose 1
      logging_level 7
      ts2phc.pulsewidth 100000000
      ts2phc.nmea_serialport $gnss_serialport
      leapfile  /usr/share/zoneinfo/leap-seconds.list
      [$iface_master]
      ts2phc.extts_polarity rising
      ts2phc.extts_correction 0
```

```
ptp4lConf: |
 [$iface_master]
 masterOnly 1
 [$iface_master_1]
 masterOnly 1
 [$iface_master_2]
 masterOnly 1
 [$iface_master_3]
 masterOnly 1
 [global]
 #
 # Default Data Set
 #
 twoStepFlag 1
 priority1 128
 priority2 128
 domainNumber 24
 #utc_offset 37
 clockClass 6
 clockAccuracy 0x27
 offsetScaledLogVariance 0xFFFF
 free_running 0
 freq_est_interval 1
 dscp_event 0
 dscp_general 0
 dataset_comparison G.8275.x
 G.8275.defaultDS.localPriority 128
 #
 # Port Data Set
 #
 logAnnounceInterval -3
 logSyncInterval -4
 logMinDelayReqInterval -4
 logMinPdelayReqInterval 0
 announceReceiptTimeout 3
 syncReceiptTimeout 0
 delayAsymmetry 0
 fault_reset_interval -4
 neighborPropDelayThresh 20000000
 masterOnly 0
 G.8275.portDS.localPriority 128
 #
 # Run time options
 #
 assume_two_step 0
 logging_level 6
 path_trace_enabled 0
 follow_up_info 0
 hybrid_e2e 0
 inhibit_multicast_service 0
 net_sync_monitor 0
 tc_spanning_tree 0
 tx_timestamp_timeout 50
 unicast_listen 0
 unicast_master_table 0
 unicast_req_duration 3600
```

```
            use_syslog 1
            verbose 0
            summary_interval -4
            kernel_leap 1
            check_fup_sync 0
            clock_class_threshold 7
            #
            # Servo Options
            #
            pi_proportional_const 0.0
            pi_integral_const 0.0
            pi_proportional_scale 0.0
            pi_proportional_exponent -0.3
            pi_proportional_norm_max 0.7
            pi_integral_scale 0.0
            pi_integral_exponent 0.4
            pi_integral_norm_max 0.3
            step_threshold 2.0
            first_step_threshold 0.00002
            clock_servo pi
            sanity_freq_limit  200000000
            ntpshm_segment 0
            #
            # Transport options
            #
            transportSpecific 0x0
            ptp_dst_mac 01:1B:19:00:00:00
            p2p_dst_mac 01:80:C2:00:00:0E
            udp_ttl 1
            udp6_scope 0x0E
            uds_address /var/run/ptp4l
            #
            # Default interface options
            #
            clock_type BC
            network_transport L2
            delay_mechanism E2E
            time_stamping hardware
            tsproc_mode filter
            delay_filter moving_median
            delay_filter_length 10
            egressLatency 0
            ingressLatency 0
            boundary_clock_jbod 0
            #
            # Clock description
            #
            productDescription ;;
            revisionData ;;
            manufacturerIdentity 00:00:00
            userDescription ;
            timeSource 0x20
        recommend:
        - profile: "grandmaster"
```

```
priority: 4
match:
- nodeLabel: "node-role.kubernetes.io/$mcp"
```

b. Create the CR by running the following command:

```
$ oc create -f grandmaster-clock-ptp-config.yaml
```

Verification

1. Check that the **PtpConfig** profile is applied to the node.

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

   ```
   $ oc get pods -n openshift-ptp -o wide
   ```

   Example output

   ```
   NAME                      READY  STATUS   RESTARTS  AGE    IP          NODE
   linuxptp-daemon-74m2g        3/3    Running  3         4d15h  10.16.230.7   compute-
   1.example.com
   ptp-operator-5f4f48d7c-x7zkf 1/1    Running  1         4d15h  10.128.1.145  compute-
   1.example.com
   ```

   b. Check that the profile is correct. Examine the logs of the **linuxptp** daemon that corresponds to the node you specified in the **PtpConfig** profile. Run the following command:

   ```
   $ oc logs linuxptp-daemon-74m2g -n openshift-ptp -c linuxptp-daemon-container
   ```

   Example output

   ```
   ts2phc[94980.334]: [ts2phc.0.config] nmea delay: 98690975 ns
   ts2phc[94980.334]: [ts2phc.0.config] ens3f0 extts index 0 at 1676577329.999999999 corr
   0 src 1676577330.901342528 diff -1
   ts2phc[94980.334]: [ts2phc.0.config] ens3f0 master offset        -1 s2 freq      -1
   ts2phc[94980.441]: [ts2phc.0.config] nmea sentence:
   GNRMC,195453.00,A,4233.24427,N,07126.64420,W,0.008,,160223,,,A,V
   phc2sys[94980.450]: [ptp4l.0.config] CLOCK_REALTIME phc offset      943 s2 freq  -
   89604 delay    504
   phc2sys[94980.512]: [ptp4l.0.config] CLOCK_REALTIME phc offset     1000 s2 freq  -
   89264 delay    474
   ```

Additional resources

- Configuring the PTP fast event notifications publisher

19.6.2.1. Grandmaster clock PtpConfig configuration reference

The following reference information describes the configuration options for the **PtpConfig** custom resource (CR) that configures the **linuxptp** services (**ptp4l**, **phc2sys**, **ts2phc**) as a grandmaster clock.

Table 19.1. PtpConfig configuration options for PTP Grandmaster clock

PtpConfig CR field	Description
plugins	Specify an array of **.exec.cmdline** options that configure the NIC for grandmaster clock operation. Grandmaster clock configuration requires certain PTP pins to be disabled. The plugin mechanism allows the PTP Operator to do automated hardware configuration. For the Intel Westport Channel NIC, when **enableDefaultConfig** is true, The PTP Operator runs a hard-coded script to do the required configuration for the NIC.
ptp4lOpts	Specify system configuration options for the **ptp4l** service. The options should not include the network interface name **-i <interface>** and service config file **-f /etc/ptp4l.conf** because the network interface name and the service config file are automatically appended.
ptp4lConf	Specify the required configuration to start **ptp4l** as a grandmaster clock. For example, the **ens2f1** interface synchronizes downstream connected devices. For grandmaster clocks, set **clockClass** to **6** and set **clockAccuracy** to **0x27**. Set **timeSource** to **0x20** for when receiving the timing signal from a Global navigation satellite system (GNSS).
tx_timestamp_timeout	Specify the maximum amount of time to wait for the transmit (TX) timestamp from the sender before discarding the data.
boundary_clock_jbod	Specify the JBOD boundary clock time delay value. This value is used to correct the time values that are passed between the network time devices.
phc2sysOpts	Specify system config options for the **phc2sys** service. If this field is empty the PTP Operator does not start the **phc2sys** service. NOTE Ensure that the network interface listed here is configured as grandmaster and is referenced as required in the **ts2phcConf** and **ptp4lConf** fields.
ptpSchedulingPolicy	Configure the scheduling policy for **ptp4l** and **phc2sys** processes. Default value is **SCHED_OTHER**. Use **SCHED_FIFO** on systems that support FIFO scheduling.
ptpSchedulingPriority	Set an integer value from 1-65 to configure FIFO priority for **ptp4l** and **phc2sys** processes when **ptpSchedulingPolicy** is set to **SCHED_FIFO**. The **ptpSchedulingPriority** field is not used when **ptpSchedulingPolicy** is set to **SCHED_OTHER**.

PtpConfig CR field	Description
ptpClockThreshold	Optional. If **ptpClockThreshold** stanza is not present, default values are used for **ptpClockThreshold** fields. Stanza shows default **ptpClockThreshold** values. **ptpClockThreshold** values configure how long after the PTP master clock is disconnected before PTP events are triggered. **holdOverTimeout** is the time value in seconds before the PTP clock event state changes to **FREERUN** when the PTP master clock is disconnected. The **maxOffsetThreshold** and **minOffsetThreshold** settings configure offset values in nanoseconds that compare against the values for **CLOCK_REALTIME** (**phc2sys**) or master offset (**ptp4l**). When the **ptp4l** or **phc2sys** offset value is outside this range, the PTP clock state is set to **FREERUN**. When the offset value is within this range, the PTP clock state is set to **LOCKED**.
ts2phcConf	Sets the configuration for the **ts2phc** command. **leapfile** is the default path to the current leap seconds definition file in the PTP Operator container image. **ts2phc.nmea_serialport** is the serial port device that is connected to the NMEA GPS clock source. When configured, the GNSS receiver is accessible on **/dev/gnss<id>**. If the host has multiple GNSS receivers, you can find the correct device by enumerating either of the following devices: • **/sys/class/net/<eth_port>/device/gnss/** • **/sys/class/gnss/gnss<id>/device/**
ts2phcOpts	Set options for the **ts2phc** command.
recommend	Specify an array of one or more **recommend** objects that define rules on how the **profile** should be applied to nodes.
.recommend.profile	Specify the **.recommend.profile** object name that is defined in the **profile** section.
.recommend.priority	Specify the **priority** with an integer value between **0** and **99**. A larger number gets lower priority, so a priority of **99** is lower than a priority of **10**. If a node can be matched with multiple profiles according to rules defined in the **match** field, the profile with the higher priority is applied to that node.
.recommend.match	Specify **.recommend.match** rules with **nodeLabel** or **nodeName** values.
.recommend.match.nodeLabel	Set **nodeLabel** with the **key** of the **node.Labels** field from the node object by using the **oc get nodes --show-labels** command. For example, **node-role.kubernetes.io/worker**.
.recommend.match.nodeName	Set **nodeName** with the value of the **node.Name** field from the node object by using the **oc get nodes** command. For example, **compute-1.example.com**.

19.6.3. Configuring linuxptp services as an ordinary clock

You can configure **linuxptp** services (**ptp4l**, **phc2sys**) as ordinary clock by creating a **PtpConfig** custom resource (CR) object.

> **NOTE**
>
> Use the following example **PtpConfig** CR as the basis to configure **linuxptp** services as an ordinary clock for your particular hardware and environment. This example CR does not configure PTP fast events. To configure PTP fast events, set appropriate values for **ptp4lOpts**, **ptp4lConf**, and **ptpClockThreshold**. **ptpClockThreshold** is required only when events are enabled. See "Configuring the PTP fast event notifications publisher" for more information.

Prerequisites

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

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

- Install the PTP Operator.

Procedure

1. Create the following **PtpConfig** CR, and then save the YAML in the **ordinary-clock-ptp-config.yaml** file.

   Example PTP ordinary clock configuration

   ```
   apiVersion: ptp.openshift.io/v1
   kind: PtpConfig
   metadata:
     name: ordinary-clock
     namespace: openshift-ptp
     annotations: {}
   spec:
     profile:
       - name: ordinary-clock
         # The interface name is hardware-specific
         interface: $interface
         ptp4lOpts: "-2 -s"
         phc2sysOpts: "-a -r -n 24"
         ptpSchedulingPolicy: SCHED_FIFO
         ptpSchedulingPriority: 10
         ptpSettings:
           logReduce: "true"
         ptp4lConf: |
           [global]
           #
           # Default Data Set
           #
           twoStepFlag 1
           slaveOnly 1
           priority1 128
           priority2 128
           domainNumber 24
           #utc_offset 37
   ```

```
clockClass 255
clockAccuracy 0xFE
offsetScaledLogVariance 0xFFFF
free_running 0
freq_est_interval 1
dscp_event 0
dscp_general 0
dataset_comparison G.8275.x
G.8275.defaultDS.localPriority 128
#
# Port Data Set
#
logAnnounceInterval -3
logSyncInterval -4
logMinDelayReqInterval -4
logMinPdelayReqInterval -4
announceReceiptTimeout 3
syncReceiptTimeout 0
delayAsymmetry 0
fault_reset_interval -4
neighborPropDelayThresh 20000000
masterOnly 0
G.8275.portDS.localPriority 128
#
# Run time options
#
assume_two_step 0
logging_level 6
path_trace_enabled 0
follow_up_info 0
hybrid_e2e 0
inhibit_multicast_service 0
net_sync_monitor 0
tc_spanning_tree 0
tx_timestamp_timeout 50
unicast_listen 0
unicast_master_table 0
unicast_req_duration 3600
use_syslog 1
verbose 0
summary_interval 0
kernel_leap 1
check_fup_sync 0
clock_class_threshold 7
#
# Servo Options
#
pi_proportional_const 0.0
pi_integral_const 0.0
pi_proportional_scale 0.0
pi_proportional_exponent -0.3
pi_proportional_norm_max 0.7
pi_integral_scale 0.0
pi_integral_exponent 0.4
pi_integral_norm_max 0.3
step_threshold 2.0
```

```
    first_step_threshold 0.00002
    max_frequency 900000000
    clock_servo pi
    sanity_freq_limit 200000000
    ntpshm_segment 0
    #
    # Transport options
    #
    transportSpecific 0x0
    ptp_dst_mac 01:1B:19:00:00:00
    p2p_dst_mac 01:80:C2:00:00:0E
    udp_ttl 1
    udp6_scope 0x0E
    uds_address /var/run/ptp4l
    #
    # Default interface options
    #
    clock_type OC
    network_transport L2
    delay_mechanism E2E
    time_stamping hardware
    tsproc_mode filter
    delay_filter moving_median
    delay_filter_length 10
    egressLatency 0
    ingressLatency 0
    boundary_clock_jbod 0
    #
    # Clock description
    #
    productDescription ;;
    revisionData ;;
    manufacturerIdentity 00:00:00
    userDescription ;
    timeSource 0xA0
  recommend:
   - profile: ordinary-clock
     priority: 4
     match:
      - nodeLabel: "node-role.kubernetes.io/$mcp"
```

Table 19.2. PTP ordinary clock CR configuration options

Custom resource field	Description
name	The name of the **PtpConfig** CR.
profile	Specify an array of one or more **profile** objects. Each profile must be uniquely named.
interface	Specify the network interface to be used by the **ptp4l** service, for example **ens787f1**.

Custom resource field	Description
ptp4lOpts	Specify system config options for the **ptp4l** service, for example **-2** to select the IEEE 802.3 network transport. The options should not include the network interface name **-i <interface>** and service config file **-f /etc/ptp4l.conf** because the network interface name and the service config file are automatically appended. Append **--summary_interval -4** to use PTP fast events with this interface.
phc2sysOpts	Specify system config options for the **phc2sys** service. If this field is empty, the PTP Operator does not start the **phc2sys** service. For Intel Columbiaville 800 Series NICs, set **phc2sysOpts** options to **-a -r -m -n 24 -N 8 -R 16**. **-m** prints messages to **stdout**. The **linuxptp-daemon DaemonSet** parses the logs and generates Prometheus metrics.
ptp4lConf	Specify a string that contains the configuration to replace the default **/etc/ptp4l.conf** file. To use the default configuration, leave the field empty.
tx_timestamp_timeout	For Intel Columbiaville 800 Series NICs, set **tx_timestamp_timeout** to **50**.
boundary_clock_jbod	For Intel Columbiaville 800 Series NICs, set **boundary_clock_jbod** to **0**.
ptpSchedulingPolicy	Scheduling policy for **ptp4l** and **phc2sys** processes. Default value is **SCHED_OTHER**. Use **SCHED_FIFO** on systems that support FIFO scheduling.
ptpSchedulingPriority	Integer value from 1-65 used to set FIFO priority for **ptp4l** and **phc2sys** processes when **ptpSchedulingPolicy** is set to **SCHED_FIFO**. The **ptpSchedulingPriority** field is not used when **ptpSchedulingPolicy** is set to **SCHED_OTHER**.
ptpClockThreshold	Optional. If **ptpClockThreshold** is not present, default values are used for the **ptpClockThreshold** fields. **ptpClockThreshold** configures how long after the PTP master clock is disconnected before PTP events are triggered. **holdOverTimeout** is the time value in seconds before the PTP clock event state changes to **FREERUN** when the PTP master clock is disconnected. The **maxOffsetThreshold** and **minOffsetThreshold** settings configure offset values in nanoseconds that compare against the values for **CLOCK_REALTIME** (**phc2sys**) or master offset (**ptp4l**). When the **ptp4l** or **phc2sys** offset value is outside this range, the PTP clock state is set to **FREERUN**. When the offset value is within this range, the PTP clock state is set to **LOCKED**.
recommend	Specify an array of one or more **recommend** objects that define rules on how the **profile** should be applied to nodes.

Custom resource field	Description
.recommend.profile	Specify the **.recommend.profile** object name defined in the **profile** section.
.recommend.priority	Set **.recommend.priority** to **0** for ordinary clock.
.recommend.match	Specify **.recommend.match** rules with **nodeLabel** or **nodeName** values.
.recommend.match.nodeLabel	Set **nodeLabel** with the **key** of the **node.Labels** field from the node object by using the **oc get nodes --show-labels** command. For example, **node-role.kubernetes.io/worker**.
.recommend.match.nodeName	Set **nodeName** with the value of the **node.Name** field from the node object by using the **oc get nodes** command. For example, **compute-1.example.com**.

2. Create the **PtpConfig** CR by running the following command:

```
$ oc create -f ordinary-clock-ptp-config.yaml
```

Verification

1. Check that the **PtpConfig** profile is applied to the node.

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

   ```
   $ oc get pods -n openshift-ptp -o wide
   ```

   Example output

   ```
   NAME                        READY  STATUS   RESTARTS  AGE  IP          NODE
   linuxptp-daemon-4xkbb        1/1    Running  0         43m  10.1.196.24   compute-
   0.example.com
   linuxptp-daemon-tdspf        1/1    Running  0         43m  10.1.196.25   compute-
   1.example.com
   ptp-operator-657bbb64c8-2f8sj 1/1   Running  0         43m  10.129.0.61   control-
   plane-1.example.com
   ```

   b. Check that the profile is correct. Examine the logs of the **linuxptp** daemon that corresponds to the node you specified in the **PtpConfig** profile. Run the following command:

   ```
   $ oc logs linuxptp-daemon-4xkbb -n openshift-ptp -c linuxptp-daemon-container
   ```

   Example output

```
I1115 09:41:17.117596 4143292 daemon.go:107] in applyNodePTPProfile
I1115 09:41:17.117604 4143292 daemon.go:109] updating NodePTPProfile to:
I1115 09:41:17.117607 4143292 daemon.go:110] -----------------------------------
I1115 09:41:17.117612 4143292 daemon.go:102] Profile Name: profile1
I1115 09:41:17.117616 4143292 daemon.go:102] Interface: ens787f1
I1115 09:41:17.117620 4143292 daemon.go:102] Ptp4lOpts: -2 -s
I1115 09:41:17.117623 4143292 daemon.go:102] Phc2sysOpts: -a -r -n 24
I1115 09:41:17.117626 4143292 daemon.go:116] -----------------------------------
```

Additional resources

- For more information about FIFO priority scheduling on PTP hardware, see Configuring FIFO priority scheduling for PTP hardware.

- For more information about configuring PTP fast events, see Configuring the PTP fast event notifications publisher.

19.6.4. Configuring linuxptp services as a boundary clock

You can configure the **linuxptp** services (**ptp4l**, **phc2sys**) as boundary clock by creating a **PtpConfig** custom resource (CR) object.

> **NOTE**
>
> Use the following example **PtpConfig** CR as the basis to configure **linuxptp** services as the boundary clock for your particular hardware and environment. This example CR does not configure PTP fast events. To configure PTP fast events, set appropriate values for **ptp4lOpts**, **ptp4lConf**, and **ptpClockThreshold**. **ptpClockThreshold** is used only when events are enabled. See "Configuring the PTP fast event notifications publisher" for more information.

Prerequisites

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

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

- Install the PTP Operator.

Procedure

1. Create the following **PtpConfig** CR, and then save the YAML in the **boundary-clock-ptp-config.yaml** file.

   Example PTP boundary clock configuration

   ```
   apiVersion: ptp.openshift.io/v1
   kind: PtpConfig
   metadata:
     name: boundary-clock
     namespace: openshift-ptp
     annotations: {}
   spec:
     profile:
   ```

```
- name: boundary-clock
  ptp4lOpts: "-2"
  phc2sysOpts: "-a -r -n 24"
  ptpSchedulingPolicy: SCHED_FIFO
  ptpSchedulingPriority: 10
  ptpSettings:
    logReduce: "true"
  ptp4lConf: |
    # The interface name is hardware-specific
    [$iface_slave]
    masterOnly 0
    [$iface_master_1]
    masterOnly 1
    [$iface_master_2]
    masterOnly 1
    [$iface_master_3]
    masterOnly 1
    [global]
    #
    # Default Data Set
    #
    twoStepFlag 1
    slaveOnly 0
    priority1 128
    priority2 128
    domainNumber 24
    #utc_offset 37
    clockClass 248
    clockAccuracy 0xFE
    offsetScaledLogVariance 0xFFFF
    free_running 0
    freq_est_interval 1
    dscp_event 0
    dscp_general 0
    dataset_comparison G.8275.x
    G.8275.defaultDS.localPriority 128
    #
    # Port Data Set
    #
    logAnnounceInterval -3
    logSyncInterval -4
    logMinDelayReqInterval -4
    logMinPdelayReqInterval -4
    announceReceiptTimeout 3
    syncReceiptTimeout 0
    delayAsymmetry 0
    fault_reset_interval -4
    neighborPropDelayThresh 20000000
    masterOnly 0
    G.8275.portDS.localPriority 128
    #
    # Run time options
    #
    assume_two_step 0
    logging_level 6
    path_trace_enabled 0
```

```
follow_up_info 0
hybrid_e2e 0
inhibit_multicast_service 0
net_sync_monitor 0
tc_spanning_tree 0
tx_timestamp_timeout 50
unicast_listen 0
unicast_master_table 0
unicast_req_duration 3600
use_syslog 1
verbose 0
summary_interval 0
kernel_leap 1
check_fup_sync 0
clock_class_threshold 135
#
# Servo Options
#
pi_proportional_const 0.0
pi_integral_const 0.0
pi_proportional_scale 0.0
pi_proportional_exponent -0.3
pi_proportional_norm_max 0.7
pi_integral_scale 0.0
pi_integral_exponent 0.4
pi_integral_norm_max 0.3
step_threshold 2.0
first_step_threshold 0.00002
max_frequency 900000000
clock_servo pi
sanity_freq_limit 200000000
ntpshm_segment 0
#
# Transport options
#
transportSpecific 0x0
ptp_dst_mac 01:1B:19:00:00:00
p2p_dst_mac 01:80:C2:00:00:0E
udp_ttl 1
udp6_scope 0x0E
uds_address /var/run/ptp4l
#
# Default interface options
#
clock_type BC
network_transport L2
delay_mechanism E2E
time_stamping hardware
tsproc_mode filter
delay_filter moving_median
delay_filter_length 10
egressLatency 0
ingressLatency 0
boundary_clock_jbod 0
#
# Clock description
```

```
        #
      productDescription ;;
      revisionData ;;
      manufacturerIdentity 00:00:00
      userDescription ;
      timeSource 0xA0
  recommend:
    - profile: boundary-clock
      priority: 4
      match:
        - nodeLabel: "node-role.kubernetes.io/$mcp"
```

Table 19.3. PTP boundary clock CR configuration options

Custom resource field	Description
name	The name of the **PtpConfig** CR.
profile	Specify an array of one or more **profile** objects.
name	Specify the name of a profile object which uniquely identifies a profile object.
ptp4lOpts	Specify system config options for the **ptp4l** service. The options should not include the network interface name **-i \<interface\>** and service config file **-f /etc/ptp4l.conf** because the network interface name and the service config file are automatically appended.
ptp4lConf	Specify the required configuration to start **ptp4l** as boundary clock. For example, **ens1f0** synchronizes from a grandmaster clock and **ens1f3** synchronizes connected devices.
\<interface_1\>	The interface that receives the synchronization clock.
\<interface_2\>	The interface that sends the synchronization clock.
tx_timestamp_timeout	For Intel Columbiaville 800 Series NICs, set **tx_timestamp_timeout** to **50**.
boundary_clock_jbod	For Intel Columbiaville 800 Series NICs, ensure **boundary_clock_jbod** is set to **0**. For Intel Fortville X710 Series NICs, ensure **boundary_clock_jbod** is set to **1**.
phc2sysOpts	Specify system config options for the **phc2sys** service. If this field is empty, the PTP Operator does not start the **phc2sys** service.
ptpSchedulingPolicy	Scheduling policy for ptp4l and phc2sys processes. Default value is **SCHED_OTHER**. Use **SCHED_FIFO** on systems that support FIFO scheduling.

Custom resource field	Description
ptpSchedulingPrio rity	Integer value from 1–65 used to set FIFO priority for **ptp4l** and **phc2sys** processes when **ptpSchedulingPolicy** is set to **SCHED_FIFO**. The **ptpSchedulingPriority** field is not used when **ptpSchedulingPolicy** is set to **SCHED_OTHER**.
ptpClockThreshold	Optional. If **ptpClockThreshold** is not present, default values are used for the **ptpClockThreshold** fields. **ptpClockThreshold** configures how long after the PTP master clock is disconnected before PTP events are triggered. **holdOverTimeout** is the time value in seconds before the PTP clock event state changes to **FREERUN** when the PTP master clock is disconnected. The **maxOffsetThreshold** and **minOffsetThreshold** settings configure offset values in nanoseconds that compare against the values for **CLOCK_REALTIME** (**phc2sys**) or master offset (**ptp4l**). When the **ptp4l** or **phc2sys** offset value is outside this range, the PTP clock state is set to **FREERUN**. When the offset value is within this range, the PTP clock state is set to **LOCKED**.
recommend	Specify an array of one or more **recommend** objects that define rules on how the **profile** should be applied to nodes.
.recommend.profil e	Specify the **.recommend.profile** object name defined in the **profile** section.
.recommend.priorit y	Specify the **priority** with an integer value between **0** and **99**. A larger number gets lower priority, so a priority of **99** is lower than a priority of **10**. If a node can be matched with multiple profiles according to rules defined in the **match** field, the profile with the higher priority is applied to that node.
.recommend.matc h	Specify **.recommend.match** rules with **nodeLabel** or **nodeName** values.
.recommend.matc h.nodeLabel	Set **nodeLabel** with the **key** of the **node.Labels** field from the node object by using the **oc get nodes --show-labels** command. For example, **node-role.kubernetes.io/worker**.
.recommend.matc h.nodeName	Set **nodeName** with the value of the **node.Name** field from the node object by using the **oc get nodes** command. For example, **compute-1.example.com**.

2. Create the CR by running the following command:

```
$ oc create -f boundary-clock-ptp-config.yaml
```

Verification

1. Check that the **PtpConfig** profile is applied to the node.

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

   ```
   $ oc get pods -n openshift-ptp -o wide
   ```

   Example output

   ```
   NAME                        READY  STATUS   RESTARTS  AGE  IP          NODE
   linuxptp-daemon-4xkbb       1/1    Running  0         43m  10.1.196.24    compute-
   0.example.com
   linuxptp-daemon-tdspf       1/1    Running  0         43m  10.1.196.25    compute-
   1.example.com
   ptp-operator-657bbb64c8-2f8sj  1/1    Running  0         43m  10.129.0.61    control-
   plane-1.example.com
   ```

   b. Check that the profile is correct. Examine the logs of the **linuxptp** daemon that corresponds to the node you specified in the **PtpConfig** profile. Run the following command:

   ```
   $ oc logs linuxptp-daemon-4xkbb -n openshift-ptp -c linuxptp-daemon-container
   ```

   Example output

   ```
   I1115 09:41:17.117596 4143292 daemon.go:107] in applyNodePTPProfile
   I1115 09:41:17.117604 4143292 daemon.go:109] updating NodePTPProfile to:
   I1115 09:41:17.117607 4143292 daemon.go:110] ----------------------------------
   I1115 09:41:17.117612 4143292 daemon.go:102] Profile Name: profile1
   I1115 09:41:17.117616 4143292 daemon.go:102] Interface:
   I1115 09:41:17.117620 4143292 daemon.go:102] Ptp4lOpts: -2
   I1115 09:41:17.117623 4143292 daemon.go:102] Phc2sysOpts: -a -r -n 24
   I1115 09:41:17.117626 4143292 daemon.go:116] ----------------------------------
   ```

Additional resources

- For more information about FIFO priority scheduling on PTP hardware, see Configuring FIFO priority scheduling for PTP hardware.

- For more information about configuring PTP fast events, see Configuring the PTP fast event notifications publisher.

19.6.5. Configuring linuxptp services as boundary clocks for dual NIC hardware

IMPORTANT

Precision Time Protocol (PTP) hardware with dual NIC configured as boundary clocks is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

You can configure the **linuxptp** services (**ptp4l**, **phc2sys**) as boundary clocks for dual NIC hardware by creating a **PtpConfig** custom resource (CR) object for each NIC.

Dual NIC hardware allows you to connect each NIC to the same upstream leader clock with separate **ptp4l** instances for each NIC feeding the downstream clocks.

Prerequisites

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

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

- Install the PTP Operator.

Procedure

1. Create two separate **PtpConfig** CRs, one for each NIC, using the reference CR in "Configuring linuxptp services as a boundary clock" as the basis for each CR. For example:

   a. Create **boundary-clock-ptp-config-nic1.yaml**, specifying values for **phc2sysOpts**:

   ```
   apiVersion: ptp.openshift.io/v1
   kind: PtpConfig
   metadata:
     name: boundary-clock-ptp-config-nic1
     namespace: openshift-ptp
   spec:
     profile:
     - name: "profile1"
       ptp4lOpts: "-2 --summary_interval -4"
       ptp4lConf: | ❶
         [ens5f1]
         masterOnly 1
         [ens5f0]
         masterOnly 0
         ...
       phc2sysOpts: "-a -r -m -n 24 -N 8 -R 16" ❷
   ```

   ❶ Specify the required interfaces to start **ptp4l** as a boundary clock. For example, **ens5f0** synchronizes from a grandmaster clock and **ens5f1** synchronizes connected devices.

   ❷ Required **phc2sysOpts** values. **-m** prints messages to **stdout**. The **linuxptp-daemon DaemonSet** parses the logs and generates Prometheus metrics.

b. Create **boundary-clock-ptp-config-nic2.yaml**, removing the **phc2sysOpts** field altogether to disable the **phc2sys** service for the second NIC:

```
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: boundary-clock-ptp-config-nic2
  namespace: openshift-ptp
spec:
  profile:
  - name: "profile2"
    ptp4lOpts: "-2 --summary_interval -4"
    ptp4lConf: |
      [ens7f1]
      masterOnly 1
      [ens7f0]
      masterOnly 0
...
```

1 Specify the required interfaces to start **ptp4l** as a boundary clock on the second NIC.
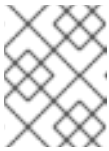
> **NOTE**
>
> You must completely remove the **phc2sysOpts** field from the second **PtpConfig** CR to disable the **phc2sys** service on the second NIC.

2. Create the dual NIC **PtpConfig** CRs by running the following commands:

   a. Create the CR that configures PTP for the first NIC:

   ```
   $ oc create -f boundary-clock-ptp-config-nic1.yaml
   ```

   b. Create the CR that configures PTP for the second NIC:

   ```
   $ oc create -f boundary-clock-ptp-config-nic2.yaml
   ```

Verification

- Check that the PTP Operator has applied the **PtpConfig** CRs for both NICs. Examine the logs for the **linuxptp** daemon corresponding to the node that has the dual NIC hardware installed. For example, run the following command:

  ```
  $ oc logs linuxptp-daemon-cvgr6 -n openshift-ptp -c linuxptp-daemon-container
  ```

Example output

```
ptp4l[80828.335]: [ptp4l.1.config] master offset          5 s2 freq   -5727 path delay       519
ptp4l[80828.343]: [ptp4l.0.config] master offset         -5 s2 freq  -10607 path delay       533
phc2sys[80828.390]: [ptp4l.0.config] CLOCK_REALTIME phc offset        1 s2 freq  -87239
delay    539
```

19.6.6. Intel Columbiaville E800 series NIC as PTP ordinary clock reference

The following table describes the changes that you must make to the reference PTP configuration in order to use Intel Columbiaville E800 series NICs as ordinary clocks. Make the changes in a **PtpConfig** custom resource (CR) that you apply to the cluster.

Table 19.4. Recommended PTP settings for Intel Columbiaville NIC

PTP configuration	Recommended setting
phc2sysOpts	**-a -r -m -n 24 -N 8 -R 16**
tx_timestamp_timeout	**50**
boundary_clock_jbod	**0**

> **NOTE**
>
> For **phc2sysOpts**, **-m** prints messages to **stdout**. The **linuxptp-daemon DaemonSet** parses the logs and generates Prometheus metrics.

Additional resources

- For a complete example CR that configures **linuxptp** services as an ordinary clock with PTP fast events, see Configuring linuxptp services as ordinary clock .

19.6.7. Configuring FIFO priority scheduling for PTP hardware

In telco or other deployment configurations that require low latency performance, PTP daemon threads run in a constrained CPU footprint alongside the rest of the infrastructure components. By default, PTP threads run with the **SCHED_OTHER** policy. Under high load, these threads might not get the scheduling latency they require for error-free operation.

To mitigate against potential scheduling latency errors, you can configure the PTP Operator **linuxptp** services to allow threads to run with a **SCHED_FIFO** policy. If **SCHED_FIFO** is set for a **PtpConfig** CR, then **ptp4l** and **phc2sys** will run in the parent container under **chrt** with a priority set by the **ptpSchedulingPriority** field of the **PtpConfig** CR.

> **NOTE**
>
> Setting **ptpSchedulingPolicy** is optional, and is only required if you are experiencing latency errors.

Procedure

1. Edit the **PtpConfig** CR profile:

   ```
   $ oc edit PtpConfig -n openshift-ptp
   ```

2. Change the **ptpSchedulingPolicy** and **ptpSchedulingPriority** fields:

   ```
   apiVersion: ptp.openshift.io/v1
   ```

```
kind: PtpConfig
metadata:
  name: <ptp_config_name>
  namespace: openshift-ptp
...
spec:
  profile:
  - name: "profile1"
...
    ptpSchedulingPolicy: SCHED_FIFO 1
    ptpSchedulingPriority: 10 2
```

1   Scheduling policy for **ptp4l** and **phc2sys** processes. Use **SCHED_FIFO** on systems that support FIFO scheduling.

2   Required. Sets the integer value 1–65 used to configure FIFO priority for **ptp4l** and **phc2sys** processes.

3. Save and exit to apply the changes to the **PtpConfig** CR.

Verification

1. Get the name of the **linuxptp-daemon** pod and corresponding node where the **PtpConfig** CR has been applied:

   ```
   $ oc get pods -n openshift-ptp -o wide
   ```

   Example output

   ```
   NAME                      READY  STATUS   RESTARTS AGE    IP          NODE
   linuxptp-daemon-gmv2n     3/3    Running  0        1d17h  10.1.196.24  compute-
   0.example.com
   linuxptp-daemon-lgm55     3/3    Running  0        1d17h  10.1.196.25  compute-
   1.example.com
   ptp-operator-3r4dcvf7f4-zndk7  1/1    Running  0        1d7h   10.129.0.61  control-plane-
   1.example.com
   ```

2. Check that the **ptp4l** process is running with the updated **chrt** FIFO priority:

   ```
   $ oc -n openshift-ptp logs linuxptp-daemon-lgm55 -c linuxptp-daemon-container|grep chrt
   ```

   Example output

   ```
   I1216 19:24:57.091872 1600715 daemon.go:285] /bin/chrt -f 65 /usr/sbin/ptp4l -f
   /var/run/ptp4l.0.config -2  --summary_interval -4 -m
   ```

19.6.8. Configuring log filtering for linuxptp services

The **linuxptp** daemon generates logs that you can use for debugging purposes. In telco or other deployment configurations that feature a limited storage capacity, these logs can add to the storage demand.

To reduce the number log messages, you can configure the **PtpConfig** custom resource (CR) to exclude log messages that report the **master offset** value. The **master offset** log message reports the difference between the current node's clock and the master clock in nanoseconds.

Prerequisites

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

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

- Install the PTP Operator.

Procedure

1. Edit the **PtpConfig** CR:

   ```
   $ oc edit PtpConfig -n openshift-ptp
   ```

2. In **spec.profile**, add the **ptpSettings.logReduce** specification and set the value to **true**:

   ```
   apiVersion: ptp.openshift.io/v1
   kind: PtpConfig
   metadata:
     name: <ptp_config_name>
     namespace: openshift-ptp
   ...
   spec:
     profile:
     - name: "profile1"
   ...
       ptpSettings:
         logReduce: "true"
   ```

   > **NOTE**
   >
   > For debugging purposes, you can revert this specification to **False** to include the master offset messages.

3. Save and exit to apply the changes to the **PtpConfig** CR.

Verification

1. Get the name of the **linuxptp-daemon** pod and corresponding node where the **PtpConfig** CR has been applied:

   ```
   $ oc get pods -n openshift-ptp -o wide
   ```

   Example output

   ```
   NAME                      READY  STATUS   RESTARTS AGE    IP          NODE
   linuxptp-daemon-gmv2n          3/3    Running  0        1d17h  10.1.196.24  compute-
   0.example.com
   linuxptp-daemon-lgm55          3/3    Running  0        1d17h  10.1.196.25  compute-
   ```

1.example.com
ptp-operator-3r4dcvf7f4-zndk7 1/1   Running 0   1d7h   10.129.0.61   control-plane-
1.example.com

2. Verify that master offset messages are excluded from the logs by running the following
   command:

   ```
   $ oc -n openshift-ptp logs <linux_daemon_container> -c linuxptp-daemon-container | grep
   "master offset" 1
   ```

   1   <linux_daemon_container> is the name of the **linuxptp-daemon** pod, for example
   linuxptp-daemon-gmv2n.

   When you configure the **logReduce** specification, this command does not report any instances
   of **master offset** in the logs of the **linuxptp** daemon.

19.7. TROUBLESHOOTING COMMON PTP OPERATOR ISSUES

Troubleshoot common problems with the PTP Operator by performing the following steps.

Prerequisites

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

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

- Install the PTP Operator on a bare-metal cluster with hosts that support PTP.

Procedure

1. Check the Operator and operands are successfully deployed in the cluster for the configured
   nodes.

   ```
   $ oc get pods -n openshift-ptp -o wide
   ```

   Example output

   ```
   NAME                     READY  STATUS   RESTARTS  AGE    IP          NODE
   linuxptp-daemon-lmvgn        3/3    Running  0       4d17h  10.1.196.24  compute-
   0.example.com
   linuxptp-daemon-qhfg7        3/3    Running  0       4d17h  10.1.196.25  compute-
   1.example.com
   ptp-operator-6b8dcbf7f4-zndk7  1/1    Running  0       5d7h   10.129.0.61  control-plane-
   1.example.com
   ```
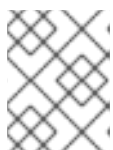
   > **NOTE**
   >
   > When the PTP fast event bus is enabled, the number of ready **linuxptp-daemon**
   > pods is **3/3**. If the PTP fast event bus is not enabled, **2/2** is displayed.

2. Check that supported hardware is found in the cluster.

```
$ oc -n openshift-ptp get nodeptpdevices.ptp.openshift.io
```

Example output

```
NAME                          AGE
control-plane-0.example.com       10d
control-plane-1.example.com       10d
compute-0.example.com            10d
compute-1.example.com            10d
compute-2.example.com            10d
```

3. Check the available PTP network interfaces for a node:

```
$ oc -n openshift-ptp get nodeptpdevices.ptp.openshift.io <node_name> -o yaml
```

where:

<node_name>

Specifies the node you want to query, for example, **compute-0.example.com**.

Example output

```
apiVersion: ptp.openshift.io/v1
kind: NodePtpDevice
metadata:
  creationTimestamp: "2021-09-14T16:52:33Z"
  generation: 1
  name: compute-0.example.com
  namespace: openshift-ptp
  resourceVersion: "177400"
  uid: 30413db0-4d8d-46da-9bef-737bacd548fd
spec: {}
status:
  devices:
  - name: eno1
  - name: eno2
  - name: eno3
  - name: eno4
  - name: enp5s0f0
  - name: enp5s0f1
```

4. Check that the PTP interface is successfully synchronized to the primary clock by accessing the **linuxptp-daemon** pod for the corresponding node.

   a. Get the name of the **linuxptp-daemon** pod and corresponding node you want to troubleshoot by running the following command:

   ```
   $ oc get pods -n openshift-ptp -o wide
   ```

   Example output

   ```
   NAME                      READY  STATUS  RESTARTS  AGE  IP        NODE
   ```

```
linuxptp-daemon-lmvgn          3/3    Running  0       4d17h  10.1.196.24   compute-
0.example.com
linuxptp-daemon-qhfg7          3/3    Running  0       4d17h  10.1.196.25   compute-
1.example.com
ptp-operator-6b8dcbf7f4-zndk7  1/1    Running  0       5d7h   10.129.0.61   control-
plane-1.example.com
```

b. Remote shell into the required **linuxptp-daemon** container:

```
$ oc rsh -n openshift-ptp -c linuxptp-daemon-container <linux_daemon_container>
```

where:

<linux_daemon_container>

is the container you want to diagnose, for example **linuxptp-daemon-lmvgn**.

c. In the remote shell connection to the **linuxptp-daemon** container, use the PTP
Management Client (**pmc**) tool to diagnose the network interface. Run the following **pmc**
command to check the sync status of the PTP device, for example **ptp4l**.

```
# pmc -u -f /var/run/ptp4l.0.config -b 0 'GET PORT_DATA_SET'
```

Example output when the node is successfully synced to the primary clock

```
sending: GET PORT_DATA_SET
  40a6b7.fffe.166ef0-1 seq 0 RESPONSE MANAGEMENT PORT_DATA_SET
    portIdentity           40a6b7.fffe.166ef0-1
    portState              SLAVE
    logMinDelayReqInterval  -4
    peerMeanPathDelay      0
    logAnnounceInterval    -3
    announceReceiptTimeout  3
    logSyncInterval        -4
    delayMechanism         1
    logMinPdelayReqInterval -4
    versionNumber          2
```

19.7.1. Collecting Precision Time Protocol (PTP) Operator data

You can use the **oc adm must-gather** CLI command to collect information about your cluster, including
features and objects associated with Precision Time Protocol (PTP) Operator.

Prerequisites

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

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

- You have installed the PTP Operator.

Procedure

- To collect PTP Operator data with **must-gather**, you must specify the PTP Operator **must-gather** image.

```
$ oc adm must-gather --image=registry.redhat.io/openshift4/ptp-must-gather-rhel8:v4.13
```

19.8. PTP HARDWARE FAST EVENT NOTIFICATIONS FRAMEWORK

Cloud native applications such as virtual RAN (vRAN) require access to notifications about hardware timing events that are critical to the functioning of the overall network. PTP clock synchronization errors can negatively affect the performance and reliability of your low-latency application, for example, a vRAN application running in a distributed unit (DU).

19.8.1. About PTP and clock synchronization error events

Loss of PTP synchronization is a critical error for a RAN network. If synchronization is lost on a node, the radio might be shut down and the network Over the Air (OTA) traffic might be shifted to another node in the wireless network. Fast event notifications mitigate against workload errors by allowing cluster nodes to communicate PTP clock sync status to the vRAN application running in the DU.

Event notifications are available to vRAN applications running on the same DU node. A publish-subscribe REST API passes events notifications to the messaging bus. Publish-subscribe messaging, or pub-sub messaging, is an asynchronous service-to-service communication architecture where any message published to a topic is immediately received by all of the subscribers to the topic.

The PTP Operator generates fast event notifications for every PTP-capable network interface. You can access the events by using a **cloud-event-proxy** sidecar container over an HTTP or Advanced Message Queuing Protocol (AMQP) message bus.
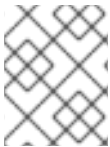
> **NOTE**
>
> PTP fast event notifications are available for network interfaces configured to use PTP ordinary clocks or PTP boundary clocks.
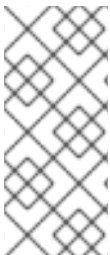
> **NOTE**
>
> HTTP transport is the default transport for PTP and bare-metal events. Use HTTP transport instead of AMQP for PTP and bare-metal events where possible. AMQ Interconnect is EOL from 30 June 2024. Extended life cycle support (ELS) for AMQ Interconnect ends 29 November 2029. For more information see, Red Hat AMQ Interconnect support status.

19.8.2. About the PTP fast event notifications framework

Use the Precision Time Protocol (PTP) fast event notifications framework to subscribe cluster applications to PTP events that the bare-metal cluster node generates.
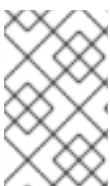
> **NOTE**
>
> The fast events notifications framework uses a REST API for communication. The REST API is based on the *O-RAN O-Cloud Notification API Specification for Event Consumers 3.0* that is available from O-RAN ALLIANCE Specifications.

The framework consists of a publisher, subscriber, and an AMQ or HTTP messaging protocol to handle communications between the publisher and subscriber applications. Applications run the **cloud-event-proxy** container in a sidecar pattern to subscribe to PTP events. The **cloud-event-proxy** sidecar

container can access the same resources as the primary application container without using any of the resources of the primary application and with no significant latency.

> **NOTE**
>
> HTTP transport is the default transport for PTP and bare-metal events. Use HTTP transport instead of AMQP for PTP and bare-metal events where possible. AMQ Interconnect is EOL from 30 June 2024. Extended life cycle support (ELS) for AMQ Interconnect ends 29 November 2029. For more information see, Red Hat AMQ Interconnect support status.

Figure 19.1. Overview of PTP fast events
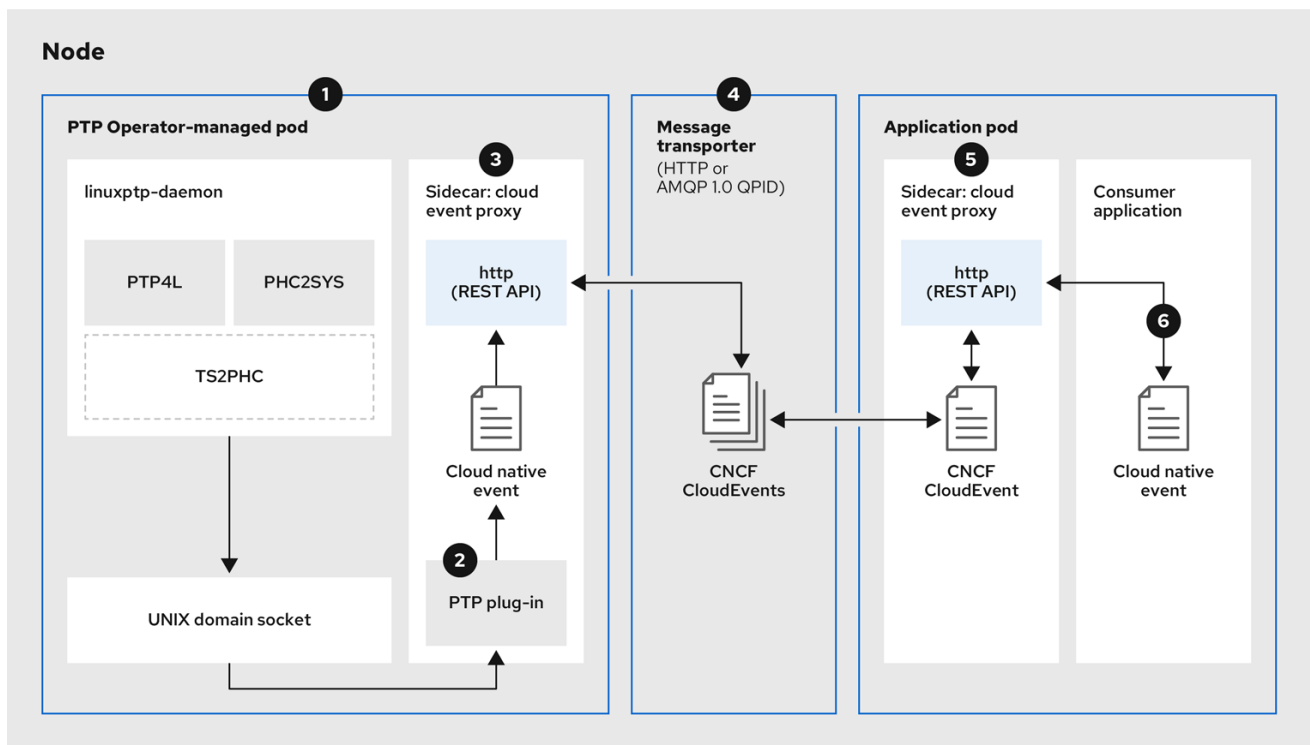
1 Event is generated on the cluster host

linuxptp-daemon in the PTP Operator-managed pod runs as a Kubernetes **DaemonSet** and manages the various **linuxptp** processes (**ptp4l**, **phc2sys**, and optionally for grandmaster clocks, **ts2phc**). The **linuxptp-daemon** passes the event to the UNIX domain socket.

2 Event is passed to the cloud-event-proxy sidecar

The PTP plugin reads the event from the UNIX domain socket and passes it to the **cloud-event-proxy** sidecar in the PTP Operator-managed pod. **cloud-event-proxy** delivers the event from the Kubernetes infrastructure to Cloud-Native Network Functions (CNFs) with low latency.

3 Event is persisted

The **cloud-event-proxy** sidecar in the PTP Operator-managed pod processes the event and publishes the cloud-native event by using a REST API.

4 Message is transported

The message transporter transports the event to the **cloud-event-proxy** sidecar in the application pod over HTTP or AMQP 1.0 QPID.

5 Event is available from the REST API

The **cloud-event-proxy** sidecar in the Application pod processes the event and makes it available by using the REST API.

6 Consumer application requests a subscription and receives the subscribed event

The consumer application sends an API request to the **cloud-event-proxy** sidecar in the application pod to create a PTP events subscription. The **cloud-event-proxy** sidecar creates an AMQ or HTTP messaging listener protocol for the resource specified in the subscription.

The **cloud-event-proxy** sidecar in the application pod receives the event from the PTP Operator-managed pod, unwraps the cloud events object to retrieve the data, and posts the event to the consumer application. The consumer application listens to the address specified in the resource qualifier and receives and processes the PTP event.

19.8.3. Configuring the PTP fast event notifications publisher

To start using PTP fast event notifications for a network interface in your cluster, you must enable the fast event publisher in the PTP Operator **PtpOperatorConfig** custom resource (CR) and configure **ptpClockThreshold** values in a **PtpConfig** CR that you create.

Prerequisites

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

- You have logged in as a user with **cluster-admin** privileges.

- You have installed the PTP Operator.

Procedure

1. Modify the default PTP Operator config to enable PTP fast events.

    a. Save the following YAML in the **ptp-operatorconfig.yaml** file:

    ```
    apiVersion: ptp.openshift.io/v1
    kind: PtpOperatorConfig
    metadata:
      name: default
      namespace: openshift-ptp
    spec:
      daemonNodeSelector:
        node-role.kubernetes.io/worker: ""
      ptpEventConfig:
        enableEventPublisher: true 1
    ```

    1 Set **enableEventPublisher** to **true** to enable PTP fast event notifications.
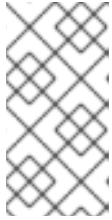
> **NOTE**
>
> In OpenShift Container Platform 4.13 or later, you do not need to set the
> **spec.ptpEventConfig.transportHost** field in the **PtpOperatorConfig** resource
> when you use HTTP transport for PTP events. Set **transportHost** only when you
> use AMQP transport for PTP events.

a. Update the **PtpOperatorConfig** CR:

```
$ oc apply -f ptp-operatorconfig.yaml
```

2. Create a **PtpConfig** custom resource (CR) for the PTP enabled interface, and set the required values for **ptpClockThreshold** and **ptp4lOpts**. The following YAML illustrates the required values that you must set in the **PtpConfig** CR:

```
spec:
  profile:
  - name: "profile1"
    interface: "enp5s0f0"
    ptp4lOpts: "-2 -s --summary_interval -4"     1
    phc2sysOpts: "-a -r -m -n 24 -N 8 -R 16"     2
    ptp4lConf: ""     3
    ptpClockThreshold:     4
      holdOverTimeout: 5
      maxOffsetThreshold: 100
      minOffsetThreshold: -100
```

1 Append **--summary_interval -4** to use PTP fast events.

2 Required **phc2sysOpts** values. **-m** prints messages to **stdout**. The **linuxptp-daemon DaemonSet** parses the logs and generates Prometheus metrics.

3 Specify a string that contains the configuration to replace the default **/etc/ptp4l.conf** file. To use the default configuration, leave the field empty.

4 Optional. If the **ptpClockThreshold** stanza is not present, default values are used for the **ptpClockThreshold** fields. The stanza shows default **ptpClockThreshold** values. The **ptpClockThreshold** values configure how long after the PTP master clock is disconnected before PTP events are triggered. **holdOverTimeout** is the time value in seconds before the PTP clock event state changes to **FREERUN** when the PTP master clock is disconnected. The **maxOffsetThreshold** and **minOffsetThreshold** settings configure offset values in nanoseconds that compare against the values for **CLOCK_REALTIME** (**phc2sys**) or master offset (**ptp4l**). When the **ptp4l** or **phc2sys** offset value is outside this range, the PTP clock state is set to **FREERUN**. When the offset value is within this range, the PTP clock state is set to **LOCKED**.

Additional resources

- For a complete example CR that configures **linuxptp** services as an ordinary clock with PTP fast events, see Configuring linuxptp services as ordinary clock .

19.8.4. Migrating consumer applications to use HTTP transport for PTP or bare-metal events

If you have previously deployed PTP or bare-metal events consumer applications, you need to update the applications to use HTTP message transport.

Prerequisites

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

- You have logged in as a user with **cluster-admin** privileges.

- You have updated the PTP Operator or Bare Metal Event Relay to version 4.13+ which uses HTTP transport by default.

Procedure

1. Update your events consumer application to use HTTP transport. Set the **http-event-publishers** variable for the cloud event sidecar deployment.
   For example, in a cluster with PTP events configured, the following YAML snippet illustrates a cloud event sidecar deployment:

   ```
   containers:
     - name: cloud-event-sidecar
       image: cloud-event-sidecar
       args:
         - "--metrics-addr=127.0.0.1:9091"
         - "--store-path=/store"
         - "--transport-host=consumer-events-subscription-service.cloud-events.svc.cluster.local:9043"
         - "--http-event-publishers=ptp-event-publisher-service-NODE_NAME.openshift-ptp.svc.cluster.local:9043" 1
         - "--api-port=8089"
   ```

   1 The PTP Operator automatically resolves **NODE_NAME** to the host that is generating the PTP events. For example, **compute-1.example.com**.

   In a cluster with bare-metal events configured, set the **http-event-publishers** field to **hw-event-publisher-service.openshift-bare-metal-events.svc.cluster.local:9043** in the cloud event sidecar deployment CR.

2. Deploy the **consumer-events-subscription-service** service alongside the events consumer application. For example:

   ```
   apiVersion: v1
   kind: Service
   metadata:
     annotations:
       prometheus.io/scrape: "true"
       service.alpha.openshift.io/serving-cert-secret-name: sidecar-consumer-secret
     name: consumer-events-subscription-service
     namespace: cloud-events
     labels:
       app: consumer-service
   spec:
     ports:
       - name: sub-port
   ```

```
    port: 9043
  selector:
    app: consumer
  clusterIP: None
  sessionAffinity: None
  type: ClusterIP
```

19.8.5. Installing the AMQ messaging bus

To pass PTP fast event notifications between publisher and subscriber on a node, you can install and configure an AMQ messaging bus to run locally on the node. To use AMQ messaging, you must install the AMQ Interconnect Operator.

> **NOTE**
>
> HTTP transport is the default transport for PTP and bare-metal events. Use HTTP transport instead of AMQP for PTP and bare-metal events where possible. AMQ Interconnect is EOL from 30 June 2024. Extended life cycle support (ELS) for AMQ Interconnect ends 29 November 2029. For more information see, Red Hat AMQ Interconnect support status.

Prerequisites

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

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

Procedure

- Install the AMQ Interconnect Operator to its own **amq-interconnect** namespace. See Adding the Red Hat Integration – AMQ Interconnect Operator.

Verification

1. Check that the AMQ Interconnect Operator is available and the required pods are running:

   ```
   $ oc get pods -n amq-interconnect
   ```

   Example output

   ```
   NAME                             READY   STATUS    RESTARTS   AGE
   amq-interconnect-645db76c76-k8ghs    1/1     Running   0          23h
   interconnect-operator-5cb5fc7cc-4v7qm   1/1     Running   0          23h
   ```

2. Check that the required **linuxptp-daemon** PTP event producer pods are running in the **openshift-ptp** namespace.

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

   Example output

```
NAME                  READY   STATUS    RESTARTS     AGE
linuxptp-daemon-2t78p  3/3    Running   0            12h
linuxptp-daemon-k8n88  3/3    Running   0            12h
```

19.8.6. Subscribing DU applications to PTP events REST API reference

Use the PTP event notifications REST API to subscribe a distributed unit (DU) application to the PTP events that are generated on the parent node.

Subscribe applications to PTP events by using the resource address **/cluster/node/<node_name>/ptp**, where **<node_name>** is the cluster node running the DU application.

Deploy your **cloud-event-consumer** DU application container and **cloud-event-proxy** sidecar container in a separate DU application pod. The **cloud-event-consumer** DU application subscribes to the **cloud-event-proxy** container in the application pod.

Use the following API endpoints to subscribe the **cloud-event-consumer** DU application to PTP events posted by the **cloud-event-proxy** container at **http://localhost:8089/api/ocloudNotifications/v1/** in the DU application pod:

- **/api/ocloudNotifications/v1/subscriptions**

  - **POST**: Creates a new subscription

  - **GET**: Retrieves a list of subscriptions

- **/api/ocloudNotifications/v1/subscriptions/<subscription_id>**

  - **GET**: Returns details for the specified subscription ID

- **/api/ocloudNotifications/v1/health**

  - **GET**: Returns the health status of **ocloudNotifications** API

- **api/ocloudNotifications/v1/publishers**

  - **GET**: Returns an array of **os-clock-sync-state**, **ptp-clock-class-change**, and **lock-state** messages for the cluster node

- **/api/ocloudnotifications/v1/<resource_address>/CurrentState**

  - **GET**: Returns the current state of one the following event types: **os-clock-sync-state**, **ptp-clock-class-change**, or **lock-state** events

> **NOTE**
>
> **9089** is the default port for the **cloud-event-consumer** container deployed in the application pod. You can configure a different port for your DU application as required.

19.8.6.1. api/ocloudNotifications/v1/subscriptions

HTTP method
GET api/ocloudNotifications/v1/subscriptions

Description

Returns a list of subscriptions. If subscriptions exist, a **200 OK** status code is returned along with the list of subscriptions.

Example API response

```
[
 {
  "id": "75b1ad8f-c807-4c23-acf5-56f4b7ee3826",
  "endpointUri": "http://localhost:9089/event",
  "uriLocation": "http://localhost:8089/api/ocloudNotifications/v1/subscriptions/75b1ad8f-c807-4c23-
acf5-56f4b7ee3826",
  "resource": "/cluster/node/compute-1.example.com/ptp"
 }
]
```

HTTP method
POST api/ocloudNotifications/v1/subscriptions

Description
Creates a new subscription. If a subscription is successfully created, or if it already exists, a **201 Created** status code is returned.

Table 19.5. Query parameters

Parameter	Type
subscription	data

Example payload

```
{
  "uriLocation": "http://localhost:8089/api/ocloudNotifications/v1/subscriptions",
  "resource": "/cluster/node/compute-1.example.com/ptp"
}
```

19.8.6.2. api/ocloudNotifications/v1/subscriptions/<subscription_id>

HTTP method
GET api/ocloudNotifications/v1/subscriptions/<subscription_id>

Description
Returns details for the subscription with ID **<subscription_id>**

Table 19.6. Query parameters

Parameter	Type
<subscription_id>	string

Example API response

```
{
```

```
    "id":"48210fb3-45be-4ce0-aa9b-41a0e58730ab",
    "endpointUri": "http://localhost:9089/event",
    "uriLocation":"http://localhost:8089/api/ocloudNotifications/v1/subscriptions/48210fb3-45be-4ce0-
aa9b-41a0e58730ab",
    "resource":"/cluster/node/compute-1.example.com/ptp"
}
```

19.8.6.3. api/ocloudNotifications/v1/health/

HTTP method
GET api/ocloudNotifications/v1/health/

Description
Returns the health status for the **ocloudNotifications** REST API.

Example API response

```
OK
```

19.8.6.4. api/ocloudNotifications/v1/publishers

HTTP method
GET api/ocloudNotifications/v1/publishers

Description
Returns an array of **os-clock-sync-state**, **ptp-clock-class-change**, and **lock-state** details for the cluster node. The system generates notifications when the relevant equipment state changes.

- **os-clock-sync-state** notifications describe the host operating system clock synchronization state. Can be in **LOCKED** or **FREERUN** state.

- **ptp-clock-class-change** notifications describe the current state of the PTP clock class.

- **lock-state** notifications describe the current status of the PTP equipment lock state. Can be in **LOCKED**, **HOLDOVER** or **FREERUN** state.

Example API response

```
[
  {
    "id": "0fa415ae-a3cf-4299-876a-589438bacf75",
    "endpointUri": "http://localhost:9085/api/ocloudNotifications/v1/dummy",
    "uriLocation": "http://localhost:9085/api/ocloudNotifications/v1/publishers/0fa415ae-a3cf-4299-
876a-589438bacf75",
    "resource": "/cluster/node/compute-1.example.com/sync/sync-status/os-clock-sync-state"
  },
  {
    "id": "28cd82df-8436-4f50-bbd9-7a9742828a71",
    "endpointUri": "http://localhost:9085/api/ocloudNotifications/v1/dummy",
    "uriLocation": "http://localhost:9085/api/ocloudNotifications/v1/publishers/28cd82df-8436-4f50-
bbd9-7a9742828a71",
    "resource": "/cluster/node/compute-1.example.com/sync/ptp-status/ptp-clock-class-change"
  },
  {
    "id": "44aa480d-7347-48b0-a5b0-e0af01fa9677",
```

```
    "endpointUri": "http://localhost:9085/api/ocloudNotifications/v1/dummy",
    "uriLocation": "http://localhost:9085/api/ocloudNotifications/v1/publishers/44aa480d-7347-48b0-
a5b0-e0af01fa9677",
    "resource": "/cluster/node/compute-1.example.com/sync/ptp-status/lock-state"
  }
]
```

You can find **os-clock-sync-state**, **ptp-clock-class-change** and **lock-state** events in the logs for the
cloud-event-proxy container. For example:

```
$ oc logs -f linuxptp-daemon-cvgr6 -n openshift-ptp -c cloud-event-proxy
```

Example os-clock-sync-state event

```
{
  "id":"c8a784d1-5f4a-4c16-9a81-a3b4313affe5",
  "type":"event.sync.sync-status.os-clock-sync-state-change",
  "source":"/cluster/compute-1.example.com/ptp/CLOCK_REALTIME",
  "dataContentType":"application/json",
  "time":"2022-05-06T15:31:23.906277159Z",
  "data":{
    "version":"v1",
    "values":[
      {
        "resource":"/sync/sync-status/os-clock-sync-state",
        "dataType":"notification",
        "valueType":"enumeration",
        "value":"LOCKED"
      },
      {
        "resource":"/sync/sync-status/os-clock-sync-state",
        "dataType":"metric",
        "valueType":"decimal64.3",
        "value":"-53"
      }
    ]
  }
}
```

Example ptp-clock-class-change event

```
{
  "id":"69eddb52-1650-4e56-b325-86d44688d02b",
  "type":"event.sync.ptp-status.ptp-clock-class-change",
  "source":"/cluster/compute-1.example.com/ptp/ens2fx/master",
  "dataContentType":"application/json",
  "time":"2022-05-06T15:31:23.147100033Z",
  "data":{
    "version":"v1",
    "values":[
      {
        "resource":"/sync/ptp-status/ptp-clock-class-change",
        "dataType":"metric",
        "valueType":"decimal64.3",
```

```
      "value":"135"
    }
  ]
}
}
```

Example lock-state event

```
{
  "id":"305ec18b-1472-47b3-aadd-8f37933249a9",
  "type":"event.sync.ptp-status.ptp-state-change",
  "source":"/cluster/compute-1.example.com/ptp/ens2fx/master",
  "dataContentType":"application/json",
  "time":"2022-05-06T15:31:23.467684081Z",
  "data":{
    "version":"v1",
    "values":[
      {
        "resource":"/sync/ptp-status/lock-state",
        "dataType":"notification",
        "valueType":"enumeration",
        "value":"LOCKED"
      },
      {
        "resource":"/sync/ptp-status/lock-state",
        "dataType":"metric",
        "valueType":"decimal64.3",
        "value":"62"
      }
    ]
  }
}
```

19.8.6.5. /api/ocloudnotifications/v1/<resource_address>/CurrentState

HTTP method
GET api/ocloudNotifications/v1/cluster/node/<node_name>/sync/ptp-status/lock-state/CurrentState

GET api/ocloudNotifications/v1/cluster/node/<node_name>/sync/sync-status/os-clock-sync-state/CurrentState

GET api/ocloudNotifications/v1/cluster/node/<node_name>/sync/ptp-status/ptp-clock-class-change/CurrentState

Description
Configure the **CurrentState** API endpoint to return the current state of the **os-clock-sync-state**, **ptp-clock-class-change**, or **lock-state** events for the cluster node.

- **os-clock-sync-state** notifications describe the host operating system clock synchronization state. Can be in **LOCKED** or **FREERUN** state.

- **ptp-clock-class-change** notifications describe the current state of the PTP clock class.

- **lock-state** notifications describe the current status of the PTP equipment lock state. Can be in **LOCKED**, **HOLDOVER** or **FREERUN** state.

Table 19.7. Query parameters

Parameter	Type
<resource_address>	string

Example lock-state API response

```
{
  "id": "c1ac3aa5-1195-4786-84f8-da0ea4462921",
  "type": "event.sync.ptp-status.ptp-state-change",
  "source": "/cluster/node/compute-1.example.com/sync/ptp-status/lock-state",
  "dataContentType": "application/json",
  "time": "2023-01-10T02:41:57.094981478Z",
  "data": {
    "version": "v1",
    "values": [
      {
        "resource": "/cluster/node/compute-1.example.com/ens5fx/master",
        "dataType": "notification",
        "valueType": "enumeration",
        "value": "LOCKED"
      },
      {
        "resource": "/cluster/node/compute-1.example.com/ens5fx/master",
        "dataType": "metric",
        "valueType": "decimal64.3",
        "value": "29"
      }
    ]
  }
}
```

Example os-clock-sync-state API response

```
{
  "specversion": "0.3",
  "id": "4f51fe99-feaa-4e66-9112-66c5c9b9afcb",
  "source": "/cluster/node/compute-1.example.com/sync/sync-status/os-clock-sync-state",
  "type": "event.sync.sync-status.os-clock-sync-state-change",
  "subject": "/cluster/node/compute-1.example.com/sync/sync-status/os-clock-sync-state",
  "datacontenttype": "application/json",
  "time": "2022-11-29T17:44:22.202Z",
  "data": {
    "version": "v1",
    "values": [
      {
        "resource": "/cluster/node/compute-1.example.com/CLOCK_REALTIME",
        "dataType": "notification",
        "valueType": "enumeration",
        "value": "LOCKED"
```

```
    },
    {
      "resource": "/cluster/node/compute-1.example.com/CLOCK_REALTIME",
      "dataType": "metric",
      "valueType": "decimal64.3",
      "value": "27"
    }
  ]
 }
}
```

Example ptp-clock-class-change API response

```
{
  "id": "064c9e67-5ad4-4afb-98ff-189c6aa9c205",
  "type": "event.sync.ptp-status.ptp-clock-class-change",
  "source": "/cluster/node/compute-1.example.com/sync/ptp-status/ptp-clock-class-change",
  "dataContentType": "application/json",
  "time": "2023-01-10T02:41:56.785673989Z",
  "data": {
    "version": "v1",
    "values": [
      {
        "resource": "/cluster/node/compute-1.example.com/ens5fx/master",
        "dataType": "metric",
        "valueType": "decimal64.3",
        "value": "165"
      }
    ]
  }
}
```

19.8.7. Monitoring PTP fast event metrics

You can monitor PTP fast events metrics from cluster nodes where the **linuxptp-daemon** is running.
You can also monitor PTP fast event metrics in the OpenShift Container Platform web console by using
the preconfigured and self-updating Prometheus monitoring stack.

Prerequisites

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

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

- Install and configure the PTP Operator on a node with PTP-capable hardware.

Procedure

1. Check for exposed PTP metrics on any node where the **linuxptp-daemon** is running. For
   example, run the following command:

   ```
   $ curl http://<node_name>:9091/metrics
   ```

   Example output

```
# HELP openshift_ptp_clock_state 0 = FREERUN, 1 = LOCKED, 2 = HOLDOVER
# TYPE openshift_ptp_clock_state gauge
openshift_ptp_clock_state{iface="ens1fx",node="compute-1.example.com",process="ptp4l"}
1
openshift_ptp_clock_state{iface="ens3fx",node="compute-1.example.com",process="ptp4l"}
1
openshift_ptp_clock_state{iface="ens5fx",node="compute-1.example.com",process="ptp4l"}
1
openshift_ptp_clock_state{iface="ens7fx",node="compute-1.example.com",process="ptp4l"}
1
# HELP openshift_ptp_delay_ns
# TYPE openshift_ptp_delay_ns gauge
openshift_ptp_delay_ns{from="master",iface="ens1fx",node="compute-
1.example.com",process="ptp4l"} 842
openshift_ptp_delay_ns{from="master",iface="ens3fx",node="compute-
1.example.com",process="ptp4l"} 480
openshift_ptp_delay_ns{from="master",iface="ens5fx",node="compute-
1.example.com",process="ptp4l"} 584
openshift_ptp_delay_ns{from="master",iface="ens7fx",node="compute-
1.example.com",process="ptp4l"} 482
openshift_ptp_delay_ns{from="phc",iface="CLOCK_REALTIME",node="compute-
1.example.com",process="phc2sys"} 547
# HELP openshift_ptp_offset_ns
# TYPE openshift_ptp_offset_ns gauge
openshift_ptp_offset_ns{from="master",iface="ens1fx",node="compute-
1.example.com",process="ptp4l"} -2
openshift_ptp_offset_ns{from="master",iface="ens3fx",node="compute-
1.example.com",process="ptp4l"} -44
openshift_ptp_offset_ns{from="master",iface="ens5fx",node="compute-
1.example.com",process="ptp4l"} -8
openshift_ptp_offset_ns{from="master",iface="ens7fx",node="compute-
1.example.com",process="ptp4l"} 3
openshift_ptp_offset_ns{from="phc",iface="CLOCK_REALTIME",node="compute-
1.example.com",process="phc2sys"} 12
```

2. To view the PTP event in the OpenShift Container Platform web console, copy the name of the PTP metric you want to query, for example, **openshift_ptp_offset_ns**.

3. In the OpenShift Container Platform web console, click **Observe → Metrics**.

4. Paste the PTP metric name into the **Expression** field, and click **Run queries**.

Additional resources

- Managing metrics

CHAPTER 20. DEVELOPING PTP EVENTS CONSUMER APPLICATIONS

When developing consumer applications that make use of Precision Time Protocol (PTP) events on a bare-metal cluster node, you need to deploy your consumer application and a **cloud-event-proxy** container in a separate application pod. The **cloud-event-proxy** container receives the events from the PTP Operator pod and passes it to the consumer application. The consumer application subscribes to the events posted in the **cloud-event-proxy** container by using a REST API.

For more information about deploying PTP events applications, see About the PTP fast event notifications framework.

> **NOTE**
>
> The following information provides general guidance for developing consumer applications that use PTP events. A complete events consumer application example is outside the scope of this information.

20.1. PTP EVENTS CONSUMER APPLICATION REFERENCE

PTP event consumer applications require the following features:

1. A web service running with a **POST** handler to receive the cloud native PTP events JSON payload

2. A **createSubscription** function to subscribe to the PTP events producer

3. A **getCurrentState** function to poll the current state of the PTP events producer

The following example Go snippets illustrate these requirements:

Example PTP events consumer server function in Go

```go
func server() {
  http.HandleFunc("/event", getEvent)
  http.ListenAndServe("localhost:8989", nil)
}

func getEvent(w http.ResponseWriter, req *http.Request) {
  defer req.Body.Close()
  bodyBytes, err := io.ReadAll(req.Body)
  if err != nil {
    log.Errorf("error reading event %v", err)
  }
  e := string(bodyBytes)
  if e != "" {
    processEvent(bodyBytes)
    log.Infof("received event %s", string(bodyBytes))
  } else {
    w.WriteHeader(http.StatusNoContent)
  }
}
```

Example PTP events createSubscription function in Go

```go
import (
"github.com/redhat-cne/sdk-go/pkg/pubsub"
"github.com/redhat-cne/sdk-go/pkg/types"
v1pubsub "github.com/redhat-cne/sdk-go/v1/pubsub"
)

// Subscribe to PTP events using REST API
s1,_:=createsubscription("/cluster/node/<node_name>/sync/sync-status/os-clock-sync-state") ①
s2,_:=createsubscription("/cluster/node/<node_name>/sync/ptp-status/ptp-clock-class-change")
s3,_:=createsubscription("/cluster/node/<node_name>/sync/ptp-status/lock-state")

// Create PTP event subscriptions POST
func createSubscription(resourceAddress string) (sub pubsub.PubSub, err error) {
 var status int
   apiPath:= "/api/ocloudNotifications/v1/"
   localAPIAddr:=localhost:8989 // vDU service API address
   apiAddr:= "localhost:8089" // event framework API address

 subURL := &types.URI{URL: url.URL{Scheme: "http",
  Host: apiAddr
  Path: fmt.Sprintf("%s%s", apiPath, "subscriptions")}}
 endpointURL := &types.URI{URL: url.URL{Scheme: "http",
  Host: localAPIAddr,
  Path: "event"}}

 sub = v1pubsub.NewPubSub(endpointURL, resourceAddress)
 var subB []byte

 if subB, err = json.Marshal(&sub); err == nil {
  rc := restclient.New()
  if status, subB = rc.PostWithReturn(subURL, subB); status != http.StatusCreated {
   err = fmt.Errorf("error in subscription creation api at %s, returned status %d", subURL, status)
  } else {
   err = json.Unmarshal(subB, &sub)
  }
 } else {
  err = fmt.Errorf("failed to marshal subscription for %s", resourceAddress)
 }
 return
}
```

① Replace **<node_name>** with the FQDN of the node that is generating the PTP events. For example, **compute-1.example.com**.

Example PTP events consumer getCurrentState function in Go

```go
//Get PTP event state for the resource
func getCurrentState(resource string) {
 //Create publisher
 url := &types.URI{URL: url.URL{Scheme: "http",
   Host: localhost:8989,
   Path: fmt.SPrintf("/api/ocloudNotifications/v1/%s/CurrentState",resource}}
 rc := restclient.New()
 status, event := rc.Get(url)
```

```
  if status != http.StatusOK {
    log.Errorf("CurrentState:error %d from url %s, %s", status, url.String(), event)
  } else {
    log.Debugf("Got CurrentState: %s ", event)
  }
}
```

20.2. REFERENCE CLOUD-EVENT-PROXY DEPLOYMENT AND SERVICE CRS

Use the following example **cloud-event-proxy** deployment and subscriber service CRs as a reference when deploying your PTP events consumer application.

> **NOTE**
>
> HTTP transport is the default transport for PTP and bare-metal events. Use HTTP transport instead of AMQP for PTP and bare-metal events where possible. AMQ Interconnect is EOL from 30 June 2024. Extended life cycle support (ELS) for AMQ Interconnect ends 29 November 2029. For more information see, Red Hat AMQ Interconnect support status.

Reference cloud-event-proxy deployment with HTTP transport

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: event-consumer-deployment
  namespace: <namespace>
  labels:
    app: consumer
spec:
  replicas: 1
  selector:
    matchLabels:
      app: consumer
  template:
    metadata:
      labels:
        app: consumer
    spec:
      serviceAccountName: sidecar-consumer-sa
      containers:
        - name: event-subscriber
          image: event-subscriber-app
        - name: cloud-event-proxy-as-sidecar
          image: openshift4/ose-cloud-event-proxy
          args:
            - "--metrics-addr=127.0.0.1:9091"
            - "--store-path=/store"
            - "--transport-host=consumer-events-subscription-service.cloud-events.svc.cluster.local:9043"
            - "--http-event-publishers=ptp-event-publisher-service-NODE_NAME.openshift-ptp.svc.cluster.local:9043"
            - "--api-port=8089"
          env:
```

```
      - name: NODE_NAME
        valueFrom:
          fieldRef:
            fieldPath: spec.nodeName
      - name: NODE_IP
        valueFrom:
          fieldRef:
            fieldPath: status.hostIP
        volumeMounts:
          - name: pubsubstore
            mountPath: /store
    ports:
      - name: metrics-port
        containerPort: 9091
      - name: sub-port
        containerPort: 9043
    volumes:
      - name: pubsubstore
        emptyDir: {}
```

Reference cloud-event-proxy deployment with AMQ transport

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: cloud-event-proxy-sidecar
  namespace: cloud-events
  labels:
    app: cloud-event-proxy
spec:
  selector:
    matchLabels:
      app: cloud-event-proxy
  template:
    metadata:
      labels:
        app: cloud-event-proxy
    spec:
      nodeSelector:
        node-role.kubernetes.io/worker: ""
      containers:
        - name: cloud-event-sidecar
          image: openshift4/ose-cloud-event-proxy
          args:
            - "--metrics-addr=127.0.0.1:9091"
            - "--store-path=/store"
            - "--transport-host=amqp://router.router.svc.cluster.local"
            - "--api-port=8089"
          env:
            - name: <node_name>
              valueFrom:
                fieldRef:
                  fieldPath: spec.nodeName
            - name: <node_ip>
              valueFrom:
                fieldRef:
```

```
        fieldPath: status.hostIP
    volumeMounts:
      - name: pubsubstore
        mountPath: /store
    ports:
      - name: metrics-port
        containerPort: 9091
      - name: sub-port
        containerPort: 9043
    volumes:
      - name: pubsubstore
        emptyDir: {}
```

Reference cloud-event-proxy subscriber service

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    prometheus.io/scrape: "true"
    service.alpha.openshift.io/serving-cert-secret-name: sidecar-consumer-secret
  name: consumer-events-subscription-service
  namespace: cloud-events
  labels:
    app: consumer-service
spec:
  ports:
    - name: sub-port
      port: 9043
  selector:
    app: consumer
  clusterIP: None
  sessionAffinity: None
  type: ClusterIP
```

20.3. PTP EVENTS AVAILABLE FROM THE CLOUD-EVENT-PROXY SIDECAR REST API

PTP events consumer applications can poll the PTP events producer for the following PTP timing events.

Table 20.1. PTP events available from the cloud-event-proxy sidecar

Resource URI	Description
/cluster/node/<node_name>/sync/ptp-status/lock-state	Describes the current status of the PTP equipment lock state. Can be in **LOCKED**, **HOLDOVER**, or **FREERUN** state.
/cluster/node/<node_name>/sync/sync-status/os-clock-sync-state	Describes the host operating system clock synchronization state. Can be in **LOCKED** or **FREERUN** state.

Resource URI	Description
/cluster/node/<node_name>/sync/ptp-status/ptp-clock-class-change	Describes the current state of the PTP clock class.

20.4. SUBSCRIBING THE CONSUMER APPLICATION TO PTP EVENTS

Before the PTP events consumer application can poll for events, you need to subscribe the application to the event producer.

20.4.1. Subscribing to PTP lock-state events

To create a subscription for PTP **lock-state** events, send a **POST** action to the cloud event API at **http://localhost:8081/api/ocloudNotifications/v1/subscriptions** with the following payload:

```
{
"endpointUri": "http://localhost:8989/event",
"resource": "/cluster/node/<node_name>/sync/ptp-status/lock-state",
}
```

Example response

```
{
"id": "e23473d9-ba18-4f78-946e-401a0caeff90",
"endpointUri": "http://localhost:8989/event",
"uriLocation": "http://localhost:8089/api/ocloudNotifications/v1/subscriptions/e23473d9-ba18-4f78-946e-401a0caeff90",
"resource": "/cluster/node/<node_name>/sync/ptp-status/lock-state",
}
```

20.4.2. Subscribing to PTP os-clock-sync-state events

To create a subscription for PTP **os-clock-sync-state** events, send a **POST** action to the cloud event API at **http://localhost:8081/api/ocloudNotifications/v1/subscriptions** with the following payload:

```
{
"endpointUri": "http://localhost:8989/event",
"resource": "/cluster/node/<node_name>/sync/sync-status/os-clock-sync-state",
}
```

Example response

```
{
"id": "e23473d9-ba18-4f78-946e-401a0caeff90",
"endpointUri": "http://localhost:8989/event",
"uriLocation": "http://localhost:8089/api/ocloudNotifications/v1/subscriptions/e23473d9-ba18-4f78-946e-401a0caeff90",
"resource": "/cluster/node/<node_name>/sync/sync-status/os-clock-sync-state",
}
```

20.4.3. Subscribing to PTP ptp-clock-class-change events

To create a subscription for PTP **ptp-clock-class-change** events, send a **POST** action to the cloud event API at **http://localhost:8081/api/ocloudNotifications/v1/subscriptions** with the following payload:

```
{
"endpointUri": "http://localhost:8989/event",
"resource": "/cluster/node/<node_name>/sync/ptp-status/ptp-clock-class-change",
}
```

Example response

```
{
"id": "e23473d9-ba18-4f78-946e-401a0caeff90",
"endpointUri": "http://localhost:8989/event",
"uriLocation": "http://localhost:8089/api/ocloudNotifications/v1/subscriptions/e23473d9-ba18-4f78-946e-401a0caeff90",
"resource": "/cluster/node/<node_name>/sync/ptp-status/ptp-clock-class-change",
}
```

20.5. GETTING THE CURRENT PTP CLOCK STATUS

To get the current PTP status for the node, send a **GET** action to one of the following event REST APIs:

- **http://localhost:8081/api/ocloudNotifications/v1/cluster/node/<node_name>/sync/ptp-status/lock-state/CurrentState**

- **http://localhost:8081/api/ocloudNotifications/v1/cluster/node/<node_name>/sync/sync-status/os-clock-sync-state/CurrentState**

- **http://localhost:8081/api/ocloudNotifications/v1/cluster/node/<node_name>/sync/ptp-status/ptp-clock-class-change/CurrentState**

The response is a cloud native event JSON object. For example:

Example lock-state API response

```
{
  "id": "c1ac3aa5-1195-4786-84f8-da0ea4462921",
  "type": "event.sync.ptp-status.ptp-state-change",
  "source": "/cluster/node/compute-1.example.com/sync/ptp-status/lock-state",
  "dataContentType": "application/json",
  "time": "2023-01-10T02:41:57.094981478Z",
  "data": {
   "version": "v1",
   "values": [
    {
      "resource": "/cluster/node/compute-1.example.com/ens5fx/master",
      "dataType": "notification",
      "valueType": "enumeration",
      "value": "LOCKED"
    },
    {
```

```
      "resource": "/cluster/node/compute-1.example.com/ens5fx/master",
      "dataType": "metric",
      "valueType": "decimal64.3",
      "value": "29"
    }
  ]
 }
}
```

20.6. VERIFYING THAT THE PTP EVENTS CONSUMER APPLICATION IS RECEIVING EVENTS

Verify that the **cloud-event-proxy** container in the application pod is receiving PTP events.

Prerequisites

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

- You have logged in as a user with **cluster-admin** privileges.

- You have installed and configured the PTP Operator.

Procedure

1. Get the list of active **linuxptp-daemon** pods. Run the following command:

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

   Example output

   ```
   NAME                READY  STATUS   RESTARTS  AGE
   linuxptp-daemon-2t78p  3/3    Running  0         8h
   linuxptp-daemon-k8n88  3/3    Running  0         8h
   ```

2. Access the metrics for the required consumer-side **cloud-event-proxy** container by running the following command:

   ```
   $ oc exec -it <linuxptp-daemon> -n openshift-ptp -c cloud-event-proxy -- curl
   127.0.0.1:9091/metrics
   ```

   where:

   <linuxptp-daemon>

      Specifies the pod you want to query, for example, **linuxptp-daemon-2t78p**.

   Example output

   ```
   # HELP cne_transport_connections_resets Metric to get number of connection resets
   # TYPE cne_transport_connections_resets gauge
   cne_transport_connection_reset 1
   # HELP cne_transport_receiver Metric to get number of receiver created
   # TYPE cne_transport_receiver gauge
   cne_transport_receiver{address="/cluster/node/compute-
   ```

```
1.example.com/ptp",status="active"} 2
cne_transport_receiver{address="/cluster/node/compute-
1.example.com/redfish/event",status="active"} 2
# HELP cne_transport_sender Metric to get number of sender created
# TYPE cne_transport_sender gauge
cne_transport_sender{address="/cluster/node/compute-
1.example.com/ptp",status="active"} 1
cne_transport_sender{address="/cluster/node/compute-
1.example.com/redfish/event",status="active"} 1
# HELP cne_events_ack Metric to get number of events produced
# TYPE cne_events_ack gauge
cne_events_ack{status="success",type="/cluster/node/compute-1.example.com/ptp"} 18
cne_events_ack{status="success",type="/cluster/node/compute-
1.example.com/redfish/event"} 18
# HELP cne_events_transport_published Metric to get number of events published by the
transport
# TYPE cne_events_transport_published gauge
cne_events_transport_published{address="/cluster/node/compute-
1.example.com/ptp",status="failed"} 1
cne_events_transport_published{address="/cluster/node/compute-
1.example.com/ptp",status="success"} 18
cne_events_transport_published{address="/cluster/node/compute-
1.example.com/redfish/event",status="failed"} 1
cne_events_transport_published{address="/cluster/node/compute-
1.example.com/redfish/event",status="success"} 18
# HELP cne_events_transport_received Metric to get number of events received  by the
transport
# TYPE cne_events_transport_received gauge
cne_events_transport_received{address="/cluster/node/compute-
1.example.com/ptp",status="success"} 18
cne_events_transport_received{address="/cluster/node/compute-
1.example.com/redfish/event",status="success"} 18
# HELP cne_events_api_published Metric to get number of events published by the rest
api
# TYPE cne_events_api_published gauge
cne_events_api_published{address="/cluster/node/compute-
1.example.com/ptp",status="success"} 19
cne_events_api_published{address="/cluster/node/compute-
1.example.com/redfish/event",status="success"} 19
# HELP cne_events_received Metric to get number of events received
# TYPE cne_events_received gauge
cne_events_received{status="success",type="/cluster/node/compute-1.example.com/ptp"}
18
cne_events_received{status="success",type="/cluster/node/compute-
1.example.com/redfish/event"} 18
# HELP promhttp_metric_handler_requests_in_flight Current number of scrapes being
served.
# TYPE promhttp_metric_handler_requests_in_flight gauge
promhttp_metric_handler_requests_in_flight 1
# HELP promhttp_metric_handler_requests_total Total number of scrapes by HTTP status
code.
# TYPE promhttp_metric_handler_requests_total counter
promhttp_metric_handler_requests_total{code="200"} 4
promhttp_metric_handler_requests_total{code="500"} 0
promhttp_metric_handler_requests_total{code="503"} 0
```

CHAPTER 21. EXTERNAL DNS OPERATOR

21.1. EXTERNAL DNS OPERATOR IN OPENSHIFT CONTAINER PLATFORM

The External DNS Operator deploys and manages **ExternalDNS** to provide the name resolution for services and routes from the external DNS provider to OpenShift Container Platform.

21.1.1. External DNS Operator

The External DNS Operator implements the External DNS API from the **olm.openshift.io** API group. The External DNS Operator updates services, routes, and external DNS providers.

Prerequisites

- You have installed the **yq** CLI tool.

Procedure

You can deploy the External DNS Operator on demand from the OperatorHub. Deploying the External DNS Operator creates a **Subscription** object.

1. Check the name of an install plan by running the following command:

   ```
   $ oc -n external-dns-operator get sub external-dns-operator -o yaml | yq '.status.installplan.name'
   ```

   Example output

   ```
   install-zcvlr
   ```

2. Check if the status of an install plan is **Complete** by running the following command:

   ```
   $ oc -n external-dns-operator get ip <install_plan_name> -o yaml | yq '.status.phase'
   ```

   Example output

   ```
   Complete
   ```

3. View the status of the **external-dns-operator** deployment by running the following command:

   ```
   $ oc get -n external-dns-operator deployment/external-dns-operator
   ```

   Example output

   ```
   NAME                  READY   UP-TO-DATE   AVAILABLE   AGE
   external-dns-operator   1/1      1            1          23h
   ```

21.1.2. External DNS Operator logs

You can view External DNS Operator logs by using the **oc logs** command.

Procedure

1. View the logs of the External DNS Operator by running the following command:

```
$ oc logs -n external-dns-operator deployment/external-dns-operator -c external-dns-operator
```

21.1.2.1. External DNS Operator domain name limitations

The External DNS Operator uses the TXT registry which adds the prefix for TXT records. This reduces the maximum length of the domain name for TXT records. A DNS record cannot be present without a corresponding TXT record, so the domain name of the DNS record must follow the same limit as the TXT records. For example, a DNS record of **<domain_name_from_source>** results in a TXT record of **external-dns-<record_type>-<domain_name_from_source>**.

The domain name of the DNS records generated by the External DNS Operator has the following limitations:

Record type	Number of characters
CNAME	44
Wildcard CNAME records on AzureDNS	42
A	48
Wildcard A records on AzureDNS	46

The following error appears in the External DNS Operator logs if the generated domain name exceeds any of the domain name limitations:

```
time="2022-09-02T08:53:57Z" level=error msg="Failure in zone test.example.io. [Id:
/hostedzone/Z06988883Q0H0RL6UMXXX]"
time="2022-09-02T08:53:57Z" level=error msg="InvalidChangeBatch: [FATAL problem:
DomainLabelTooLong (Domain label is too long) encountered with 'external-dns-a-hello-openshift-
aaaaaaaaaa-bbbbbbbbbb-cccccccc']\n\tstatus code: 400, request id: e54dfd5a-06c6-47b0-bcb9-
a4f7c3a4e0c6"
```

21.2. INSTALLING EXTERNAL DNS OPERATOR ON CLOUD PROVIDERS

You can install the External DNS Operator on cloud providers such as AWS, Azure, and GCP.

21.2.1. Installing the External DNS Operator

You can install the External DNS Operator by using the OpenShift Container Platform OperatorHub.

Procedure

1. Click **Operators** → **OperatorHub** in the OpenShift Container Platform web console.

2. Click **External DNS Operator**. You can use the **Filter by keyword** text box or the filter list to search for External DNS Operator from the list of Operators.

3. Select the **external-dns-operator** namespace.

4. On the **External DNS Operator** page, click **Install**.

5. On the **Install Operator** page, ensure that you selected the following options:

   a. Update the channel as **stable-v1**.

   b. Installation mode as **A specific name on the cluster**.

   c. Installed namespace as **external-dns-operator**. If namespace **external-dns-operator** does not exist, it gets created during the Operator installation.

   d. Select **Approval Strategy** as **Automatic** or **Manual**. Approval Strategy is set to **Automatic** by default.

   e. Click **Install**.

If you select **Automatic** updates, the Operator Lifecycle Manager (OLM) automatically upgrades the running instance of your Operator without any intervention.

If you select **Manual** updates, the OLM creates an update request. As a cluster administrator, you must then manually approve that update request to have the Operator updated to the new version.

Verification

Verify that the External DNS Operator shows the **Status** as **Succeeded** on the **Installed Operators** dashboard.

21.3. EXTERNAL DNS OPERATOR CONFIGURATION PARAMETERS

The External DNS Operator includes the following configuration parameters.

21.3.1. External DNS Operator configuration parameters

The External DNS Operator includes the following configuration parameters:

Parameter	Description

Parameter	Description
| **spec** | Enables the type of a cloud provider.<br><br>```
spec:
  provider:
    type: AWS ❶
    aws:
      credentials:
        name: aws-access-key ❷
```<br><br>❶ Defines available options such as AWS, GCP, Azure, and Infoblox.<br><br>❷ Defines a secret name for your cloud provider. |
| **zones** | Enables you to specify DNS zones by their domains. If you do not specify zones, the **ExternalDNS** resource discovers all of the zones present in your cloud provider account.<br><br>```
zones:
- "myzoneid" ❶
```<br><br>❶ Specifies the name of DNS zones. |
| **domains** | Enables you to specify AWS zones by their domains. If you do not specify domains, the **ExternalDNS** resource discovers all of the zones present in your cloud provider account.<br><br>```
domains:
- filterType: Include ❶
  matchType: Exact ❷
  name: "myzonedomain1.com" ❸
- filterType: Include
  matchType: Pattern ❹
  pattern: ".*\\.otherzonedomain\\.com" ❺
```<br><br>❶ Ensures that the **ExternalDNS** resource includes the domain name.<br><br>❷ Instructs **ExtrnalDNS** that the domain matching has to be exact as opposed to regular expression match.<br><br>❸ Defines the name of the domain.<br><br>❹ Sets the **regex-domain-filter** flag in the **ExternalDNS** resource. You can limit possible domains by using a Regex filter.<br><br>❺ Defines the regex pattern to be used by the **ExternalDNS** resource to filter the domains of the target zones. |
| **source** | Enables you to specify the source for the DNS records, **Service** or **Route**. |

Parameter	Description

```
source: 1
  type: Service 2
  service:
    serviceType: 3
      - LoadBalancer
      - ClusterIP
  labelFilter: 4
    matchLabels:
      external-dns.mydomain.org/publish: "yes"
  hostnameAnnotation: "Allow" 5
  fqdnTemplate:
  - "{{.Name}}.myzonedomain.com" 6
```

[1] Defines the settings for the source of DNS records.

[2] The **ExternalDNS** resource uses the **Service** type as the source for creating DNS records.

[3] Sets the **service-type-filter** flag in the **ExternalDNS** resource. The **serviceType** contains the following fields:

- **default**: **LoadBalancer**

- **expected**: **ClusterIP**

- **NodePort**

- **LoadBalancer**

- **ExternalName**

[4] Ensures that the controller considers only those resources which matches with label filter.

[5] The default value for **hostnameAnnotation** is **Ignore** which instructs **ExternalDNS** to generate DNS records using the templates specified in the field **fqdnTemplates**. When the value is **Allow** the DNS records get generated based on the value specified in the **external-dns.alpha.kubernetes.io/hostname** annotation.

[6] The External DNS Operator uses a string to generate DNS names from sources that don't define a hostname, or to add a hostname suffix when paired with the fake source.

```
source:
  type: OpenShiftRoute 1
  openshiftRouteOptions:
    routerName: default 2
    labelFilter:
      matchLabels:
        external-dns.mydomain.org/publish: "yes"
```

[1] Creates DNS records.

[2] If the source type is **OpenShiftRoute**, then you can pass the Ingress Controller name. The **ExternalDNS** resource uses the canonical name of the Ingress Controller as the target for CNAME records.

Parameter	Description

21.4. CREATING DNS RECORDS ON AWS

You can create DNS records on AWS and AWS GovCloud by using External DNS Operator.

21.4.1. Creating DNS records on an public hosted zone for AWS by using Red Hat External DNS Operator

You can create DNS records on a public hosted zone for AWS by using the Red Hat External DNS Operator. You can use the same instructions to create DNS records on a hosted zone for AWS GovCloud.

Procedure

1. Check the user. The user must have access to the **kube-system** namespace. If you don't have the credentials, as you can fetch the credentials from the **kube-system** namespace to use the cloud provider client:

   ```
   $ oc whoami
   ```

   Example output

   ```
   system:admin
   ```

2. Fetch the values from aws-creds secret present in **kube-system** namespace.

   ```
   $ export AWS_ACCESS_KEY_ID=$(oc get secrets aws-creds -n kube-system  --template=
   {{.data.aws_access_key_id}} | base64 -d)
   $ export AWS_SECRET_ACCESS_KEY=$(oc get secrets aws-creds -n kube-system  --
   template={{.data.aws_secret_access_key}} | base64 -d)
   ```

3. Get the routes to check the domain:

   ```
   $ oc get routes --all-namespaces | grep console
   ```

   Example output

   ```
   openshift-console        console         console-openshift-
   console.apps.testextdnsoperator.apacshift.support          console          https
   reencrypt/Redirect    None
   openshift-console        downloads        downloads-openshift-
   console.apps.testextdnsoperator.apacshift.support          downloads        http
   edge/Redirect        None
   ```

4. Get the list of dns zones to find the one which corresponds to the previously found route's domain:

   ```
   $ aws route53 list-hosted-zones | grep testextdnsoperator.apacshift.support
   ```

Example output

```
HOSTEDZONES terraform /hostedzone/Z02355203TNN1XXXX1J6O
testextdnsoperator.apacshift.support. 5
```

5. Create **ExternalDNS** resource for **route** source:

```
$ cat <<EOF | oc create -f -
apiVersion: externaldns.olm.openshift.io/v1beta1
kind: ExternalDNS
metadata:
  name: sample-aws 1
spec:
  domains:
  - filterType: Include 2
    matchType: Exact 3
    name: testextdnsoperator.apacshift.support 4
  provider:
    type: AWS 5
  source: 6
    type: OpenShiftRoute 7
    openshiftRouteOptions:
      routerName: default 8
EOF
```

[1]   Defines the name of external DNS resource.

[2]   By default all hosted zones are selected as potential targets. You can include a hosted zone that you need.

[3]   The matching of the target zone's domain has to be exact (as opposed to regular expression match).

[4]   Specify the exact domain of the zone you want to update. The hostname of the routes must be subdomains of the specified domain.

[5]   Defines the **AWS Route53** DNS provider.

[6]   Defines options for the source of DNS records.

[7]   Defines OpenShift **route** resource as the source for the DNS records which gets created in the previously specified DNS provider.

[8]   If the source is **OpenShiftRoute**, then you can pass the OpenShift Ingress Controller name. External DNS Operator selects the canonical hostname of that router as the target while creating CNAME record.

6. Check the records created for OCP routes using the following command:

```
$ aws route53 list-resource-record-sets --hosted-zone-id Z02355203TNN1XXXX1J6O --
query "ResourceRecordSets[?Type == 'CNAME']" | grep console
```

21.5. CREATING DNS RECORDS ON AZURE

You can create DNS records on Azure by using the External DNS Operator.

21.5.1. Creating DNS records on an Azure public DNS zone

You can create DNS records on a public DNS zone for Azure by using the External DNS Operator.

Prerequisites

- You must have administrator privileges.

- The **admin** user must have access to the **kube-system** namespace.

Procedure

1. Fetch the credentials from the **kube-system** namespace to use the cloud provider client by running the following command:

   ```
   $ CLIENT_ID=$(oc get secrets azure-credentials  -n kube-system  --template=
   {{.data.azure_client_id}} | base64 -d)
   $ CLIENT_SECRET=$(oc get secrets azure-credentials  -n kube-system  --template=
   {{.data.azure_client_secret}} | base64 -d)
   $ RESOURCE_GROUP=$(oc get secrets azure-credentials  -n kube-system  --template=
   {{.data.azure_resourcegroup}} | base64 -d)
   $ SUBSCRIPTION_ID=$(oc get secrets azure-credentials  -n kube-system  --template=
   {{.data.azure_subscription_id}} | base64 -d)
   $ TENANT_ID=$(oc get secrets azure-credentials  -n kube-system  --template=
   {{.data.azure_tenant_id}} | base64 -d)
   ```

2. Log in to Azure by running the following command:

   ```
   $ az login --service-principal -u "${CLIENT_ID}" -p "${CLIENT_SECRET}" --tenant
   "${TENANT_ID}"
   ```

3. Get a list of routes by running the following command:

   ```
   $ oc get routes --all-namespaces | grep console
   ```

   Example output

   ```
   openshift-console        console         console-openshift-
   console.apps.test.azure.example.com                     console        https   reencrypt/Redirect
   None
   openshift-console        downloads       downloads-openshift-
   console.apps.test.azure.example.com                     downloads       http    edge/Redirect
   None
   ```

4. Get a list of DNS zones by running the following command:

   ```
   $ az network dns zone list --resource-group "${RESOURCE_GROUP}"
   ```

5. Create a YAML file, for example, **external-dns-sample-azure.yaml**, that defines the
ExternalDNS object:

Example external-dns-sample-azure.yaml file

```
apiVersion: externaldns.olm.openshift.io/v1beta1
kind: ExternalDNS
metadata:
  name: sample-azure 1
spec:
  zones:
  - "/subscriptions/1234567890/resourceGroups/test-azure-xxxxx-
rg/providers/Microsoft.Network/dnszones/test.azure.example.com" 2
  provider:
    type: Azure 3
  source:
    openshiftRouteOptions: 4
      routerName: default 5
    type: OpenShiftRoute 6
```

1 Specifies the External DNS name.

2 Defines the zone ID.

3 Defines the provider type.

4 You can define options for the source of DNS records.

5 If the source type is **OpenShiftRoute**, you can pass the OpenShift Ingress Controller
name. External DNS selects the canonical hostname of that router as the target while
creating CNAME record.

6 Defines the **route** resource as the source for the Azure DNS records.

6. Check the DNS records created for OpenShift Container Platform routes by running the
following command:

```
$ az network dns record-set list -g "${RESOURCE_GROUP}" -z test.azure.example.com |
grep console
```

> **NOTE**
>
> To create records on private hosted zones on private Azure DNS, you need to
> specify the private zone under the **zones** field which populates the provider type
> to **azure-private-dns** in the **ExternalDNS** container arguments.

21.6. CREATING DNS RECORDS ON GCP

You can create DNS records on GCP by using the External DNS Operator.

21.6.1. Creating DNS records on a public managed zone for GCP

You can create DNS records on a public managed zone for GCP by using the External DNS Operator.

Prerequisites

- You must have administrator privileges.

Procedure

1. Copy the **gcp-credentials** secret in the **encoded-gcloud.json** file by running the following command:

   ```
   $ oc get secret gcp-credentials -n kube-system --template='{{$v := index .data
   "service_account.json"}}{{$v}}' | base64 -d - > decoded-gcloud.json
   ```

2. Export your Google credentials by running the following command:

   ```
   $ export GOOGLE_CREDENTIALS=decoded-gcloud.json
   ```

3. Activate your account by using the following command:

   ```
   $ gcloud auth activate-service-account  <client_email as per decoded-gcloud.json> --key-
   file=decoded-gcloud.json
   ```

4. Set your project by running the following command:

   ```
   $ gcloud config set project <project_id as per decoded-gcloud.json>
   ```

5. Get a list of routes by running the following command:

   ```
   $ oc get routes --all-namespaces | grep console
   ```

   Example output

   ```
   openshift-console       console           console-openshift-
   console.apps.test.gcp.example.com                     console        https   reencrypt/Redirect
   None
   openshift-console       downloads         downloads-openshift-
   console.apps.test.gcp.example.com                     downloads      http    edge/Redirect
   None
   ```

6. Get a list of managed zones by running the following command:

   ```
   $ gcloud dns managed-zones list | grep test.gcp.example.com
   ```

   Example output

   ```
   qe-cvs4g-private-zone test.gcp.example.com
   ```

7. Create a YAML file, for example, **external-dns-sample-gcp.yaml**, that defines the **ExternalDNS** object:

   Example external-dns-sample-gcp.yaml file

```
apiVersion: externaldns.olm.openshift.io/v1beta1
kind: ExternalDNS
metadata:
  name: sample-gcp 1
spec:
  domains:
    - filterType: Include 2
      matchType: Exact 3
      name: test.gcp.example.com 4
  provider:
    type: GCP 5
  source:
    openshiftRouteOptions: 6
      routerName: default 7
    type: OpenShiftRoute 8
```

1   Specifies the External DNS name.

2   By default, all hosted zones are selected as potential targets. You can include your hosted zone.

3   The domain of the target must match the string defined by the **name** key.

4   Specify the exact domain of the zone you want to update. The hostname of the routes must be subdomains of the specified domain.

5   Defines the provider type.

6   You can define options for the source of DNS records.

7   If the source type is **OpenShiftRoute**, you can pass the OpenShift Ingress Controller name. External DNS selects the canonical hostname of that router as the target while creating CNAME record.

8   Defines the **route** resource as the source for GCP DNS records.

8. Check the DNS records created for OpenShift Container Platform routes by running the following command:

   ```
   $ gcloud dns record-sets list --zone=qe-cvs4g-private-zone | grep console
   ```

21.7. CREATING DNS RECORDS ON INFOBLOX

You can create DNS records on Infoblox by using the External DNS Operator.

21.7.1. Creating DNS records on a public DNS zone on Infoblox

You can create DNS records on a public DNS zone on Infoblox by using the External DNS Operator.

Prerequisites

- You have access to the OpenShift CLI (**oc**).

- You have access to the Infoblox UI.

Procedure

1. Create a **secret** object with Infoblox credentials by running the following command:

   ```
   $ oc -n external-dns-operator create secret generic infoblox-credentials --from-
   literal=EXTERNAL_DNS_INFOBLOX_WAPI_USERNAME=<infoblox_username> --from-
   literal=EXTERNAL_DNS_INFOBLOX_WAPI_PASSWORD=<infoblox_password>
   ```

2. Get a list of routes by running the following command:

   ```
   $ oc get routes --all-namespaces | grep console
   ```

   Example Output

   ```
   openshift-console        console         console-openshift-console.apps.test.example.com
   console         https    reencrypt/Redirect    None
   openshift-console        downloads        downloads-openshift-
   console.apps.test.example.com                    downloads        http    edge/Redirect
   None
   ```

3. Create a YAML file, for example, **external-dns-sample-infoblox.yaml**, that defines the **ExternalDNS** object:

   Example external-dns-sample-infoblox.yaml file

   ```
   apiVersion: externaldns.olm.openshift.io/v1beta1
   kind: ExternalDNS
   metadata:
     name: sample-infoblox 1
   spec:
     provider:
       type: Infoblox 2
       infoblox:
         credentials:
           name: infoblox-credentials
         gridHost: ${INFOBLOX_GRID_PUBLIC_IP}
         wapiPort: 443
         wapiVersion: "2.3.1"
     domains:
     - filterType: Include
       matchType: Exact
       name: test.example.com
     source:
       type: OpenShiftRoute 3
       openshiftRouteOptions:
         routerName: default 4
   ```

   1 Specifies the External DNS name.

   2 Defines the provider type.

3 You can define options for the source of DNS records.

4 If the source type is **OpenShiftRoute**, you can pass the OpenShift Ingress Controller name. External DNS selects the canonical hostname of that router as the target while creating CNAME record.

4. Create the **ExternalDNS** resource on Infoblox by running the following command:

```
$ oc create -f external-dns-sample-infoblox.yaml
```

5. From the Infoblox UI, check the DNS records created for **console** routes:

   a. Click **Data Management → DNS → Zones**.

   b. Select the zone name.

21.8. CONFIGURING THE CLUSTER-WIDE PROXY ON THE EXTERNAL DNS OPERATOR

After configuring the cluster-wide proxy, the Operator Lifecycle Manager (OLM) triggers automatic updates to all of the deployed Operators with the new contents of the **HTTP_PROXY**, **HTTPS_PROXY**, and **NO_PROXY** environment variables.

21.8.1. Trusting the certificate authority of the cluster-wide proxy

You can configure the External DNS Operator to trust the certificate authority of the cluster-wide proxy.

Procedure

1. Create the config map to contain the CA bundle in the **external-dns-operator** namespace by running the following command:

```
$ oc -n external-dns-operator create configmap trusted-ca
```

2. To inject the trusted CA bundle into the config map, add the **config.openshift.io/inject-trusted-cabundle=true** label to the config map by running the following command:

```
$ oc -n external-dns-operator label cm trusted-ca config.openshift.io/inject-trusted-cabundle=true
```

3. Update the subscription of the External DNS Operator by running the following command:

```
$ oc -n external-dns-operator patch subscription external-dns-operator --type='json' -p='[{"op": "add", "path": "/spec/config", "value":{"env": [{"name":"TRUSTED_CA_CONFIGMAP_NAME","value":"trusted-ca"}]}}]'
```

Verification

- After the deployment of the External DNS Operator is completed, verify that the trusted CA environment variable is added to the **external-dns-operator** deployment by running the following command:

```
$ oc -n external-dns-operator exec deploy/external-dns-operator -c external-dns-operator --
printenv TRUSTED_CA_CONFIGMAP_NAME
```

Example output

```
trusted-ca
```

CHAPTER 22. NETWORK POLICY

22.1. ABOUT NETWORK POLICY

As a cluster administrator, you can define network policies that restrict traffic to pods in your cluster.

22.1.1. About network policy

In a cluster using a network plugin that supports Kubernetes network policy, network isolation is controlled entirely by **NetworkPolicy** objects. In OpenShift Container Platform 4.13, OpenShift SDN supports using network policy in its default network isolation mode.

> **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 or from their resident nodes.

By default, all pods in a project are accessible from other pods and network endpoints. To isolate one or more pods in a project, you can create **NetworkPolicy** objects in that project to indicate the allowed incoming connections. Project administrators can create and delete **NetworkPolicy** objects within their own project.

If a pod is matched by selectors in one or more **NetworkPolicy** objects, then the pod will accept 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: []
  ```

- Only allow connections from the OpenShift Container Platform Ingress Controller:

To make a project allow only connections from the OpenShift Container Platform Ingress Controller, add the following **NetworkPolicy** object.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-openshift-ingress
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          network.openshift.io/policy-group: ingress
  podSelector: {}
  policyTypes:
  - Ingress
```

- Only accept connections from pods within a project:
  To make pods accept connections from other pods in the same project, but reject all other connections from pods in other projects, 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 samples, you can define both **allow-same-namespace** and **allow-http-and-https** policies within the same project. Thus allowing 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.

22.1.1.1. Using the allow-from-router network policy

Use the following **NetworkPolicy** to allow external traffic regardless of the router configuration:

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-router
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          policy-group.network.openshift.io/ingress: "" 1
  podSelector: {}
  policyTypes:
  - Ingress
```

[1]   **policy-group.network.openshift.io/ingress:""** label supports both OpenShift-SDN and OVN-Kubernetes.

22.1.1.2. Using the allow-from-hostnetwork network policy

Add the following **allow-from-hostnetwork NetworkPolicy** object to direct traffic from the host network pods:

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

```
kind: NetworkPolicy
metadata:
  name: allow-from-hostnetwork
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          policy-group.network.openshift.io/host-network: ""
  podSelector: {}
  policyTypes:
  - Ingress
```

22.1.2. Optimizations for network policy with OpenShift SDN

Use a network policy to isolate pods that are differentiated from one another by labels within a namespace.

It is inefficient to apply **NetworkPolicy** objects to large numbers of individual pods in a single namespace. Pod labels do not exist at the IP address level, so a network policy generates a separate Open vSwitch (OVS) flow rule for every possible link between every pod selected with a **podSelector**.

For example, if the spec **podSelector** and the ingress **podSelector** within a **NetworkPolicy** object each match 200 pods, then 40,000 (200*200) OVS flow rules are generated. This might slow down a node.

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

- Reduce the number of OVS flow rules by using namespaces to contain groups of pods that need to be isolated.
  NetworkPolicy objects that select a whole namespace, by using the **namespaceSelector** or an empty **podSelector**, generate only a single OVS flow rule that matches the VXLAN virtual network ID (VNID) of the namespace.

- Keep the pods that do not need to be isolated in their original namespace, and move the pods that require isolation into one or more different namespaces.

- Create additional targeted cross-namespace network policies to allow the specific traffic that you do want to allow from the isolated pods.

22.1.3. 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.

22.1.4. Next steps

- Creating a network policy

- Optional: Defining a default network policy

22.1.5. Additional resources

- Projects and namespaces

- Configuring multitenant network policy

- NetworkPolicy API

22.2. CREATING A NETWORK POLICY

As a user with the **admin** role, you can create a network policy for a namespace.

22.2.1. Example NetworkPolicy object

The following annotates an example NetworkPolicy object:

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-27107 ❶
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. The policy object can only select pods in the project that defines the NetworkPolicy object.

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.

22.2.2. 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.

> **NOTE**
>
> If you log in with a user with the **cluster-admin** role, then you can create a network policy in any namespace in the cluster.

Prerequisites

- Your cluster uses a network plugin that supports **NetworkPolicy** objects, such as the OVN-Kubernetes network plugin or the OpenShift SDN network plugin with **mode: NetworkPolicy** set. This mode is the default for OpenShift SDN.

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

- You are logged in to the cluster with a user with **admin** privileges.

- 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:
  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

> **NOTE**
>
> If you log in to the web console with **cluster-admin** privileges, you have a choice of creating a network policy in any namespace in the cluster directly in YAML or from a form in the web console.

22.2.3. 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.

> **NOTE**
>
> If you log in with a user with the **cluster-admin** role, then you can create a network policy in any namespace in the cluster.

Prerequisites

- Your cluster uses a network plugin that supports **NetworkPolicy** objects, such as the OVN-Kubernetes network plugin or the OpenShift SDN network plugin with **mode: NetworkPolicy** set. This mode is the default for OpenShift SDN.

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

- You are logged in to the cluster with a user with **admin** privileges.

- 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 ❶
   ```

```
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
```

22.2.4. 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**
>
> If you log in with a user with the **cluster-admin** role, then you can create a network policy in any namespace in the cluster.

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

- Your cluster uses a network plugin that supports **NetworkPolicy** objects, such as the OVN-Kubernetes network plugin or the OpenShift SDN network plugin with **mode: NetworkPolicy** set. This mode is the default for OpenShift SDN.

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

- You are logged in to the cluster with a user with **admin** privileges.

- 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
```

This policy allows traffic from all resources, including external traffic as illustrated in the following diagram:

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

> **NOTE**
>
> If you log in with a user with the **cluster-admin** role, then you can create a network policy in any namespace in the cluster.

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

Prerequisites

- Your cluster uses a network plugin that supports **NetworkPolicy** objects, such as the OVN-Kubernetes network plugin or the OpenShift SDN network plugin with **mode: NetworkPolicy** set. This mode is the default for OpenShift SDN.

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

- You are logged in to the cluster with a user with **admin** privileges.

- 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>
```

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

NOTE

If you log in with a user with the **cluster-admin** role, then you can create a network policy in any namespace in the cluster.

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

- Your cluster uses a network plugin that supports **NetworkPolicy** objects, such as the OVN-Kubernetes network plugin or the OpenShift SDN network plugin with **mode: NetworkPolicy** set. This mode is the default for OpenShift SDN.

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

- You are logged in to the cluster with a user with **admin** privileges.

- 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>
```

22.2.7. Additional resources

- Accessing the web console

- Logging for egress firewall and network policy rules

22.3. VIEWING A NETWORK POLICY

As a user with the **admin** role, you can view a network policy for a namespace.

22.3.1. 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. The policy object can only select pods in the project that defines the NetworkPolicy object.

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.

22.3.2. Viewing network policies using the CLI

You can examine the network policies in a namespace.

> **NOTE**
>
> If you log in with a user with the **cluster-admin** role, then you can view any network policy in the cluster.

Prerequisites

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

- You are logged in to the cluster with a user with **admin** privileges.

- 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

> **NOTE**
>
> If you log in to the web console with **cluster-admin** privileges, you have a choice of
> viewing a network policy in any namespace in the cluster directly in YAML or from a form
> in the web console.

22.4. EDITING A NETWORK POLICY

As a user with the **admin** role, you can edit an existing network policy for a namespace.

22.4.1. Editing a network policy

You can edit a network policy in a namespace.

> **NOTE**
>
> If you log in with a user with the **cluster-admin** role, then you can edit a network policy in
> any namespace in the cluster.

Prerequisites

- Your cluster uses a network plugin that supports **NetworkPolicy** objects, such as the OVN-
  Kubernetes network plugin or the OpenShift SDN network plugin with **mode: NetworkPolicy**
  set. This mode is the default for OpenShift SDN.

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

- You are logged in to the cluster with a user with **admin** privileges.

- 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.

> **NOTE**
>
> If you log in to the web console with **cluster-admin** privileges, you have a choice of editing a network policy in any namespace in the cluster directly in YAML or from the policy in the web console through the **Actions** menu.

22.4.2. Example NetworkPolicy object

The following annotates an example NetworkPolicy object:

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-27107 ❶
```

```
spec:
  podSelector: 2
    matchLabels:
      app: mongodb
  ingress:
  - from:
    - podSelector: 3
        matchLabels:
          app: app
    ports: 4
    - protocol: TCP
      port: 27017
```
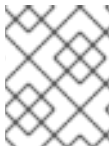
[1] The name of the NetworkPolicy object.

[2] A selector that describes the pods to which the policy applies. The policy object can only select pods in the project that defines the NetworkPolicy object.

[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.

22.4.3. Additional resources

- [Creating a network policy](#)

22.5. DELETING A NETWORK POLICY

As a user with the **admin** role, you can delete a network policy from a namespace.

22.5.1. Deleting a network policy using the CLI

You can delete a network policy in a namespace.

> **NOTE**
>
> If you log in with a user with the **cluster-admin** role, then you can delete any network policy in the cluster.

Prerequisites

- Your cluster uses a network plugin that supports **NetworkPolicy** objects, such as the OVN-Kubernetes network plugin or the OpenShift SDN network plugin with **mode: NetworkPolicy** set. This mode is the default for OpenShift SDN.

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

- You are logged in to the cluster with a user with **admin** privileges.

- 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
  ```

> **NOTE**
>
> If you log in to the web console with **cluster-admin** privileges, you have a choice of deleting a network policy in any namespace in the cluster directly in YAML or from the policy in the web console through the **Actions** menu.

22.6. DEFINING A DEFAULT NETWORK POLICY FOR PROJECTS

As a cluster administrator, you can modify the new project template to automatically include network policies when you create a new project. If you do not yet have a customized template for new projects, you must first create one.

22.6.1. Modifying the template for new projects

As a cluster administrator, you can modify the default project template so that new projects are created using your custom requirements.

To create your own custom project template:

Procedure

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

2. Generate the default project template:

   ```
   $ oc adm create-bootstrap-project-template -o yaml > template.yaml
   ```

3. Use a text editor to modify the generated **template.yaml** file by adding objects or modifying existing objects.

4. The project template must be created in the **openshift-config** namespace. Load your modified template:

   ```
   $ oc create -f template.yaml -n openshift-config
   ```

5. Edit the project configuration resource using the web console or CLI.

- Using the web console:

  i. Navigate to the **Administration → Cluster Settings** page.

  ii. Click **Configuration** to view all configuration resources.

  iii. Find the entry for **Project** and click **Edit YAML**.

- Using the CLI:

  i. Edit the **project.config.openshift.io/cluster** resource:

  ```
  $ oc edit project.config.openshift.io/cluster
  ```

6. Update the **spec** section to include the **projectRequestTemplate** and **name** parameters, and set the name of your uploaded project template. The default name is **project-request**.

   Project configuration resource with custom project template

   ```
   apiVersion: config.openshift.io/v1
   kind: Project
   metadata:
   # ...
   spec:
     projectRequestTemplate:
       name: <template_name>
   # ...
   ```

7. After you save your changes, create a new project to verify that your changes were successfully applied.

22.6.2. Adding network policies to the new project template

As a cluster administrator, you can add network policies to the default template for new projects. OpenShift Container Platform will automatically create all the **NetworkPolicy** objects specified in the template in the project.

Prerequisites

- Your cluster uses a default CNI network plugin that supports **NetworkPolicy** objects, such as the OpenShift SDN network plugin with **mode: NetworkPolicy** set. This mode is the default for OpenShift SDN.

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

- You must log in to the cluster with a user with **cluster-admin** privileges.

- You must have created a custom default project template for new projects.

Procedure

1. Edit the default template for a new project by running the following command:

   ```
   $ oc edit template <project_template> -n openshift-config
   ```

Replace **<project_template>** with the name of the default template that you configured for your cluster. The default template name is **project-request**.

2. In the template, add each **NetworkPolicy** object as an element to the **objects** parameter. The **objects** parameter accepts a collection of one or more objects.
   In the following example, the **objects** parameter collection includes several **NetworkPolicy** objects.

```
objects:
- apiVersion: networking.k8s.io/v1
  kind: NetworkPolicy
  metadata:
    name: allow-from-same-namespace
  spec:
    podSelector: {}
    ingress:
    - from:
      - podSelector: {}
- apiVersion: networking.k8s.io/v1
  kind: NetworkPolicy
  metadata:
    name: allow-from-openshift-ingress
  spec:
    ingress:
    - from:
      - namespaceSelector:
          matchLabels:
            network.openshift.io/policy-group: ingress
    podSelector: {}
    policyTypes:
    - Ingress
- apiVersion: networking.k8s.io/v1
  kind: NetworkPolicy
  metadata:
    name: allow-from-kube-apiserver-operator
  spec:
    ingress:
    - from:
      - namespaceSelector:
          matchLabels:
            kubernetes.io/metadata.name: openshift-kube-apiserver-operator
        podSelector:
          matchLabels:
            app: kube-apiserver-operator
    policyTypes:
    - Ingress
...
```

3. Optional: Create a new project to confirm that your network policy objects are created successfully by running the following commands:

   a. Create a new project:

   ```
   $ oc new-project <project>  1
   ```

1   Replace **<project>** with the name for the project you are creating.

b. Confirm that the network policy objects in the new project template exist in the new project:

```
$ oc get networkpolicy
NAME                       POD-SELECTOR  AGE
allow-from-openshift-ingress  <none>        7s
allow-from-same-namespace     <none>        7s
```

22.7. CONFIGURING MULTITENANT ISOLATION WITH NETWORK POLICY

As a cluster administrator, you can configure your network policies to provide multitenant network isolation.

NOTE

If you are using the OpenShift SDN network plugin, configuring network policies as described in this section provides network isolation similar to multitenant mode but with network policy mode set.

22.7.1. Configuring multitenant isolation by using network policy

You can configure your project to isolate it from pods and services in other project namespaces.

Prerequisites

- Your cluster uses a network plugin that supports **NetworkPolicy** objects, such as the OVN‑Kubernetes network plugin or the OpenShift SDN network plugin with **mode: NetworkPolicy** set. This mode is the default for OpenShift SDN.

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

- You are logged in to the cluster with a user with **admin** privileges.

Procedure

1. Create the following **NetworkPolicy** objects:

   a. A policy named **allow-from-openshift-ingress**.

   ```
   $ cat << EOF| oc create -f -
   apiVersion: networking.k8s.io/v1
   kind: NetworkPolicy
   metadata:
     name: allow-from-openshift-ingress
   spec:
     ingress:
     - from:
       - namespaceSelector:
           matchLabels:
             policy-group.network.openshift.io/ingress: ""
     podSelector: {}
   ```

```
policyTypes:
- Ingress
EOF
```

NOTE

policy-group.network.openshift.io/ingress: "" is the preferred namespace selector label for OpenShift SDN. You can use the **network.openshift.io/policy-group: ingress** namespace selector label, but this is a legacy label.

b. A policy named **allow-from-openshift-monitoring**:

```
$ cat << EOF| oc create -f -
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-openshift-monitoring
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          network.openshift.io/policy-group: monitoring
  podSelector: {}
  policyTypes:
  - Ingress
EOF
```

c. A policy named **allow-same-namespace**:

```
$ cat << EOF| oc create -f -
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-same-namespace
spec:
  podSelector:
  ingress:
  - from:
    - podSelector: {}
EOF
```

d. A policy named **allow-from-kube-apiserver-operator**:

```
$ cat << EOF| oc create -f -
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-kube-apiserver-operator
spec:
  ingress:
  - from:
    - namespaceSelector:
```

```
      matchLabels:
        kubernetes.io/metadata.name: openshift-kube-apiserver-operator
      podSelector:
        matchLabels:
          app: kube-apiserver-operator
    policyTypes:
    - Ingress
  EOF
```

For more details, see New **kube-apiserver-operator** webhook controller validating health of webhook.

2. Optional: To confirm that the network policies exist in your current project, enter the following command:

```
$ oc describe networkpolicy
```

Example output

```
Name:         allow-from-openshift-ingress
Namespace:    example1
Created on:   2020-06-09 00:28:17 -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:
      NamespaceSelector: network.openshift.io/policy-group: ingress
  Not affecting egress traffic
  Policy Types: Ingress


Name:         allow-from-openshift-monitoring
Namespace:    example1
Created on:   2020-06-09 00:29:57 -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:
      NamespaceSelector: network.openshift.io/policy-group: monitoring
  Not affecting egress traffic
  Policy Types: Ingress
```

22.7.2. Next steps

- Defining a default network policy

22.7.3. Additional resources

- OpenShift SDN network isolation modes

CHAPTER 23. CIDR RANGE DEFINITIONS

You must specify non-overlapping ranges for the following CIDR ranges.

> **NOTE**
>
> Machine CIDR ranges cannot be changed after creating your cluster.

> **IMPORTANT**
>
> OVN-Kubernetes, the default network provider in OpenShift Container Platform 4.11 and later, uses the **100.64.0.0/16** IP address range internally. If your cluster uses OVN-Kubernetes, do not include the **100.64.0.0/16** IP address range in any other CIDR definitions in your cluster.

23.1. MACHINE CIDR

In the Machine CIDR field, you must specify the IP address range for machines or cluster nodes.

The default is **10.0.0.0/16**. This range must not conflict with any connected networks.

23.2. SERVICE CIDR

In the Service CIDR field, you must specify the IP address range for services. The range must be large enough to accommodate your workload. The address block must not overlap with any external service accessed from within the cluster. The default is **172.30.0.0/16**.

23.3. POD CIDR

In the pod CIDR field, you must specify the IP address range for pods.

The pod CIDR is the same as the **clusterNetwork** CIDR and the cluster CIDR. The range must be large enough to accommodate your workload. The address block must not overlap with any external service accessed from within the cluster. The default is **10.128.0.0/14**. You can expand the range after cluster installation.

Additional resources

- [Cluster Network Operator Configuration](#)

- [Configuring the cluster network range](#)

23.4. HOST PREFIX

In the Host Prefix field, you must specify the subnet prefix length assigned to pods scheduled to individual machines. The host prefix determines the pod IP address pool for each machine.

For example, if the host prefix is set to **/23**, each machine is assigned a **/23** subnet from the pod CIDR address range. The default is **/23**, allowing 510 cluster nodes, and 510 pod IP addresses per node.

CHAPTER 24. AWS LOAD BALANCER OPERATOR

24.1. AWS LOAD BALANCER OPERATOR RELEASE NOTES

The AWS Load Balancer (ALB) Operator deploys and manages an instance of the **AWSLoadBalancerController** resource.

These release notes track the development of the AWS Load Balancer Operator in OpenShift Container Platform.

For an overview of the AWS Load Balancer Operator, see AWS Load Balancer Operator in OpenShift Container Platform.

> **NOTE**
>
> AWS Load Balancer Operator currently does not support AWS GovCloud.

24.1.1. AWS Load Balancer Operator 1.0.0

The AWS Load Balancer Operator is now generally available with this release. The AWS Load Balancer Operator version 1.0.0 supports the AWS Load Balancer Controller version 2.4.4.

The following advisory is available for the AWS Load Balancer Operator version 1.0.0:

- RHEA-2023:1954 Release of AWS Load Balancer Operator on OperatorHub Enhancement Advisory Update

24.1.1.1. Notable changes

- This release uses the new **v1** API version.

24.1.1.2. Bug fixes

- Previously, the controller provisioned by the AWS Load Balancer Operator did not properly use the configuration for the cluster-wide proxy. These settings are now applied appropriately to the controller. (OCPBUGS-4052, OCPBUGS-5295)

24.1.2. Earlier versions

The two earliest versions of the AWS Load Balancer Operator are available as a Technology Preview. These versions should not be used in a production cluster. For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

The following advisory is available for the AWS Load Balancer Operator version 0.2.0:

- RHEA-2022:9084 Release of AWS Load Balancer Operator on OperatorHub Enhancement Advisory Update

The following advisory is available for the AWS Load Balancer Operator version 0.0.1:

- RHEA-2022:5780 Release of AWS Load Balancer Operator on OperatorHub Enhancement Advisory Update

24.2. AWS LOAD BALANCER OPERATOR IN OPENSHIFT CONTAINER PLATFORM

The AWS Load Balancer Operator deploys and manages the AWS Load Balancer Controller. You can install the AWS Load Balancer Operator from OperatorHub by using OpenShift Container Platform web console or CLI.

24.2.1. AWS Load Balancer Operator considerations

Review the following limitations before installing and using the AWS Load Balancer Operator:

- The IP traffic mode only works on AWS Elastic Kubernetes Service (EKS). The AWS Load Balancer Operator disables the IP traffic mode for the AWS Load Balancer Controller. As a result of disabling the IP traffic mode, the AWS Load Balancer Controller cannot use the pod readiness gate.

- The AWS Load Balancer Operator adds command-line flags such as **--disable-ingress-class-annotation** and **--disable-ingress-group-name-annotation** to the AWS Load Balancer Controller. Therefore, the AWS Load Balancer Operator does not allow using the **kubernetes.io/ingress.class** and **alb.ingress.kubernetes.io/group.name** annotations in the **Ingress** resource.

24.2.2. AWS Load Balancer Operator

The AWS Load Balancer Operator can tag the public subnets if the **kubernetes.io/role/elb** tag is missing. Also, the AWS Load Balancer Operator detects the following information from the underlying AWS cloud:

- The ID of the virtual private cloud (VPC) on which the cluster hosting the Operator is deployed in.

- Public and private subnets of the discovered VPC.

The AWS Load Balancer Operator supports the Kubernetes service resource of type **LoadBalancer** by using Network Load Balancer (NLB) with the **instance** target type only.

Prerequisites

- You must have the AWS credentials secret. The credentials are used to provide subnet tagging and VPC discovery.

Procedure

1. You can deploy the AWS Load Balancer Operator on demand from OperatorHub, by creating a **Subscription** object by running the following command:

   ```
   $ oc -n aws-load-balancer-operator get sub aws-load-balancer-operator --template='{{.status.installplan.name}}{{"\n"}}'
   ```

   Example output

   ```
   install-zlfbt
   ```

2. Check if the status of an install plan is **Complete** by running the following command:

```
$ oc -n aws-load-balancer-operator get ip <install_plan_name> --template='{{.status.phase}}
{{"\n"}}'
```

Example output

```
Complete
```

3. View the status of the **aws-load-balancer-operator-controller-manager** deployment by running the following command:

```
$ oc get -n aws-load-balancer-operator deployment/aws-load-balancer-operator-controller-manager
```

Example output

```
NAME                                        READY   UP-TO-DATE  AVAILABLE  AGE
aws-load-balancer-operator-controller-manager  1/1     1           1          23h
```

24.2.3. AWS Load Balancer Operator logs

You can view the AWS Load Balancer Operator logs by using the **oc logs** command.

Procedure

- View the logs of the AWS Load Balancer Operator by running the following command:

```
$ oc logs -n aws-load-balancer-operator deployment/aws-load-balancer-operator-controller-manager -c manager
```

24.3. INSTALLING THE AWS LOAD BALANCER OPERATOR

The AWS Load Balancer Operator deploys and manages the AWS Load Balancer Controller. You can install the AWS Load Balancer Operator from the OperatorHub by using OpenShift Container Platform web console or CLI.

24.3.1. Installing the AWS Load Balancer Operator by using the web console

You can install the AWS Load Balancer Operator by using the web console.

Prerequisites

- You have logged in to the OpenShift Container Platform web console as a user with **cluster-admin** permissions.

- Your cluster is configured with AWS as the platform type and cloud provider.

- If you are using a security token service (STS) or user-provisioned infrastructure, follow the related preparation steps. For example, if you are using AWS Security Token Service, see "Preparing for the AWS Load Balancer Operator on a cluster using the AWS Security Token Service (STS)".

Procedure

1. Navigate to **Operators → OperatorHub** in the OpenShift Container Platform web console.

2. Select the **AWS Load Balancer Operator**. You can use the **Filter by keyword** text box or use the filter list to search for the AWS Load Balancer Operator from the list of Operators.

3. Select the **aws-load-balancer-operator** namespace.

4. On the **Install Operator** page, select the following options:

   a. **Update the channel** as **stable-v1**.

   b. **Installation mode** as **All namespaces on the cluster (default)**

   c. **Installed Namespace** as **aws-load-balancer-operator**. If the **aws-load-balancer-operator** namespace does not exist, it gets created during the Operator installation.

   d. Select **Update approval** as **Automatic** or **Manual**. By default, the **Update approval** is set to **Automatic**. If you select automatic updates, the Operator Lifecycle Manager (OLM) automatically upgrades the running instance of your Operator without any intervention. If you select manual updates, the OLM creates an update request. As a cluster administrator, you must then manually approve that update request to update the Operator updated to the new version.

5. Click **Install**.

Verification

- Verify that the AWS Load Balancer Operator shows the **Status** as **Succeeded** on the Installed Operators dashboard.

24.3.2. Installing the AWS Load Balancer Operator by using the CLI

You can install the AWS Load Balancer Operator by using the CLI.

Prerequisites

- You are logged in to the OpenShift Container Platform web console as a user with **cluster-admin** permissions.

- Your cluster is configured with AWS as the platform type and cloud provider.

- You are logged into the OpenShift CLI (**oc**).

Procedure

1. Create a **Namespace** object:

   a. Create a YAML file that defines the **Namespace** object:

      Example namespace.yaml file

      ```
      apiVersion: v1
      kind: Namespace
      metadata:
      ```

```
name: aws-load-balancer-operator
```

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

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

2. Create a **CredentialsRequest** object:

a. Create a YAML file that defines the **CredentialsRequest** object:

Example credentialsrequest.yaml file

```
apiVersion: cloudcredential.openshift.io/v1
kind: CredentialsRequest
metadata:
  name: aws-load-balancer-operator
  namespace: openshift-cloud-credential-operator
spec:
  providerSpec:
    apiVersion: cloudcredential.openshift.io/v1
    kind: AWSProviderSpec
    statementEntries:
     - action:
        - ec2:DescribeSubnets
       effect: Allow
       resource: "*"
     - action:
        - ec2:CreateTags
        - ec2:DeleteTags
       effect: Allow
       resource: arn:aws:ec2:*:*:subnet/*
     - action:
        - ec2:DescribeVpcs
       effect: Allow
       resource: "*"
  secretRef:
    name: aws-load-balancer-operator
    namespace: aws-load-balancer-operator
  serviceAccountNames:
    - aws-load-balancer-operator-controller-manager
```

b. Create the **CredentialsRequest** object by running the following command:

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

3. Create an **OperatorGroup** object:

a. Create a YAML file that defines the **OperatorGroup** object:

Example operatorgroup.yaml file

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
```

```
    name: aws-lb-operatorgroup
    namespace: aws-load-balancer-operator
  spec:
    upgradeStrategy: Default
```

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

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

4. Create a **Subscription** object:

   a. Create a YAML file that defines the **Subscription** object:

   Example subscription.yaml file

   ```
   apiVersion: operators.coreos.com/v1alpha1
   kind: Subscription
   metadata:
     name: aws-load-balancer-operator
     namespace: aws-load-balancer-operator
   spec:
     channel: stable-v1
     installPlanApproval: Automatic
     name: aws-load-balancer-operator
     source: qe-app-registry
     sourceNamespace: openshift-marketplace
   ```

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

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

Verification

1. Get the name of the install plan from the subscription:

   ```
   $ oc -n aws-load-balancer-operator \
       get subscription aws-load-balancer-operator \
       --template='{{.status.installplan.name}}{{"\n"}}'
   ```

2. Check the status of the install plan:

   ```
   $ oc -n aws-load-balancer-operator \
       get ip <install_plan_name> \
       --template='{{.status.phase}}{{"\n"}}'
   ```

   The output must be **Complete**.

24.4. INSTALLING THE AWS LOAD BALANCER OPERATOR ON A CLUSTER USING THE AWS SECURITY TOKEN SERVICE

You can install the AWS Load Balancer Operator on a cluster that uses STS.

The AWS Load Balancer Operator relies on the **CredentialsRequest** object to bootstrap the Operator

and the AWS Load Balancer Controller. The AWS Load Balancer Operator waits until the required secrets are created and available. The Cloud Credential Operator does not provision the secrets automatically in the STS cluster. You must set the credentials secrets manually by using the **ccoctl** binary.

If you do not want to provision credential secret by using the Cloud Credential Operator, you can configure the **AWSLoadBalancerController** instance on the STS cluster by specifying the credential secret in the AWS load Balancer Controller custom resource (CR).

24.4.1. Bootstrapping AWS Load Balancer Operator on Security Token Service cluster

Prerequisites

- You must extract and prepare the **ccoctl** binary.

Procedure

1. Create the **aws-load-balancer-operator** namespace by running the following command:

   ```
   $ oc create namespace aws-load-balancer-operator
   ```

2. Download the **CredentialsRequest** custom resource (CR) of the AWS Load Balancer Operator, and create a directory to store it by running the following command:

   ```
   $ curl --create-dirs -o <path-to-credrequests-dir>/cr.yaml
   https://raw.githubusercontent.com/openshift/aws-load-balancer-operator/main/hack/operator-credentials-request.yaml
   ```

3. Use the **ccoctl** tool to process **CredentialsRequest** objects of the AWS Load Balancer Operator, by running the following command:

   ```
   $ ccoctl aws create-iam-roles \
       --name <name> --region=<aws_region> \
       --credentials-requests-dir=<path-to-credrequests-dir> \
       --identity-provider-arn <oidc-arn>
   ```

4. Apply the secrets generated in the manifests directory of your cluster by running the following command:

   ```
   $ ls manifests/*-credentials.yaml | xargs -I{} oc apply -f {}
   ```

5. Verify that the credentials secret of the AWS Load Balancer Operator is created by running the following command:

   ```
   $ oc -n aws-load-balancer-operator get secret aws-load-balancer-operator --template='{{index .data "credentials"}}' | base64 -d
   ```

   Example output

   ```
   [default]
   sts_regional_endpoints = regional
   role_arn = arn:aws:iam::999999999999:role/aws-load-balancer-operator-aws-load-balancer-
   ```

```
operator
web_identity_token_file = /var/run/secrets/openshift/serviceaccount/token
```

24.4.2. Configuring AWS Load Balancer Operator on Security Token Service cluster by using managed CredentialsRequest objects

Prerequisites

- You must extract and prepare the **ccoctl** binary.

Procedure

1. The AWS Load Balancer Operator creates the **CredentialsRequest** object in the **openshift-cloud-credential-operator** namespace for each **AWSLoadBalancerController** custom resource (CR). You can extract and save the created **CredentialsRequest** object in a directory by running the following command:

    ```
    $ oc get credentialsrequest -n openshift-cloud-credential-operator  \
      aws-load-balancer-controller-<cr-name> -o yaml > <path-to-credrequests-dir>/cr.yaml
    ```
    1

    1 The **aws-load-balancer-controller-<cr-name>** parameter specifies the credential request name created by the AWS Load Balancer Operator. The **cr-name** specifies the name of the AWS Load Balancer Controller instance.

2. Use the **ccoctl** tool to process all **CredentialsRequest** objects in the **credrequests** directory by running the following command:

    ```
    $ ccoctl aws create-iam-roles \
        --name <name> --region=<aws_region> \
        --credentials-requests-dir=<path-to-credrequests-dir> \
        --identity-provider-arn <oidc-arn>
    ```

3. Apply the secrets generated in manifests directory to your cluster, by running the following command:

    ```
    $ ls manifests/*-credentials.yaml | xargs -I{} oc apply -f {}
    ```

4. Verify that the **aws-load-balancer-controller** pod is created:

    ```
    $ oc -n aws-load-balancer-operator get pods
    NAME                                                  READY   STATUS    RESTARTS   AGE
    aws-load-balancer-controller-cluster-9b766d6-gg82c         1/1    Running   0         137m
    aws-load-balancer-operator-controller-manager-b55ff68cc-85jzg   2/2    Running   0
    3h26m
    ```

24.4.3. Configuring the AWS Load Balancer Operator on Security Token Service cluster by using specific credentials

You can specify the credential secret by using the **spec.credentials** field in the AWS Load Balancer Controller custom resource (CR). You can use the predefined **CredentialsRequest** object of the controller to know which roles are required.

Prerequisites

- You must extract and prepare the **ccoctl** binary.

Procedure

1. Download the CredentialsRequest custom resource (CR) of the AWS Load Balancer Controller, and create a directory to store it by running the following command:

   ```
   $ curl --create-dirs -o <path-to-credrequests-dir>/cr.yaml
   https://raw.githubusercontent.com/openshift/aws-load-balancer-
   operator/main/hack/controller/controller-credentials-request.yaml
   ```

2. Use the **ccoctl** tool to process the **CredentialsRequest** object of the controller:

   ```
   $ ccoctl aws create-iam-roles \
       --name <name> --region=<aws_region> \
       --credentials-requests-dir=<path-to-credrequests-dir> \
       --identity-provider-arn <oidc-arn>
   ```

3. Apply the secrets to your cluster:

   ```
   $ ls manifests/*-credentials.yaml | xargs -I{} oc apply -f {}
   ```

4. Verify the credentials secret has been created for use by the controller:

   ```
   $ oc -n aws-load-balancer-operator get secret aws-load-balancer-controller-manual-cluster --
   template='{{index .data "credentials"}}' | base64 -d
   ```

   Example output

   ```
   [default]
       sts_regional_endpoints = regional
       role_arn = arn:aws:iam::999999999999:role/aws-load-balancer-operator-aws-load-
   balancer-controller
       web_identity_token_file = /var/run/secrets/openshift/serviceaccount/token
   ```

5. Create the **AWSLoadBalancerController** resource YAML file, for example, **sample-aws-lb-manual-creds.yaml**, as follows:

   ```
   apiVersion: networking.olm.openshift.io/v1
   kind: AWSLoadBalancerController     1
   metadata:
     name: cluster     2
   spec:
     credentials:
       name: <secret-name>     3
   ```

   1 Defines the **AWSLoadBalancerController** resource.

   2 Defines the AWS Load Balancer Controller instance name. This instance name gets added as a suffix to all related resources.

3   Specifies the secret name containing AWS credentials that the controller uses.

24.4.4. Additional resources

- [Configuring the Cloud Credential Operator utility](#)

24.5. CREATING AN INSTANCE OF THE AWS LOAD BALANCER CONTROLLER

After installing the AWS Load Balancer Operator, you can create the AWS Load Balancer Controller.

24.5.1. Creating the AWS Load Balancer Controller

You can install only a single instance of the **AWSLoadBalancerController** object in a cluster. You can create the AWS Load Balancer Controller by using CLI. The AWS Load Balancer Operator reconciles only the **cluster** named resource.

Prerequisites

- You have created the **echoserver** namespace.

- You have access to the OpenShift CLI (**oc**).

Procedure

1. Create a YAML file that defines the **AWSLoadBalancerController** object:

    Example sample-aws-lb.yaml file

    ```
    apiVersion: networking.olm.openshift.io/v1
    kind: AWSLoadBalancerController  1
    metadata:
      name: cluster  2
    spec:
      subnetTagging: Auto  3
      additionalResourceTags:  4
      - key: example.org/security-scope
        value: staging
      ingressClass: alb  5
      config:
        replicas: 2  6
      enabledAddons:  7
        - AWSWAFv2  8
    ```

    1   Defines the **AWSLoadBalancerController** object.

    2   Defines the AWS Load Balancer Controller name. This instance name gets added as a suffix to all related resources.

    3   Configures the subnet tagging method for the AWS Load Balancer Controller. The following values are valid:

- **Auto**: The AWS Load Balancer Operator determines the subnets that belong to the cluster and tags them appropriately. The Operator cannot determine the role correctly if the internal subnet tags are not present on internal subnet.

- **Manual**: You manually tag the subnets that belong to the cluster with the appropriate role tags. Use this option if you installed your cluster on user-provided infrastructure.

(4) Defines the tags used by the AWS Load Balancer Controller when it provisions AWS resources.

(5) Defines the ingress class name. The default value is **alb**.

(6) Specifies the number of replicas of the AWS Load Balancer Controller.

(7) Specifies annotations as an add-on for the AWS Load Balancer Controller.

(8) Enables the **alb.ingress.kubernetes.io/wafv2-acl-arn** annotation.

2. Create the **AWSLoadBalancerController** object by running the following command:

   ```
   $ oc create -f sample-aws-lb.yaml
   ```

3. Create a YAML file that defines the **Deployment** resource:

   Example **sample-aws-lb.yaml** file

   ```
   apiVersion: apps/v1
   kind: Deployment (1)
   metadata:
     name: <echoserver> (2)
     namespace: echoserver
   spec:
     selector:
       matchLabels:
         app: echoserver
     replicas: 3 (3)
     template:
       metadata:
         labels:
           app: echoserver
       spec:
         containers:
           - image: openshift/origin-node
             command:
              - "/bin/socat"
             args:
               - TCP4-LISTEN:8080,reuseaddr,fork
               - EXEC:/bin/bash -c \"printf \\\"HTTP/1.0 200 OK\r\n\r\n\\\"; sed -e \\\"/^\r/q\\\"\"
             imagePullPolicy: Always
             name: echoserver
             ports:
               - containerPort: 8080
   ```

(1) Defines the deployment resource.

2   Specifies the deployment name.

3   Specifies the number of replicas of the deployment.

4. Create a YAML file that defines the **Service** resource:

   Example **service-albo.yaml** file:

   ```yaml
   apiVersion: v1
   kind: Service 1
   metadata:
     name: <echoserver> 2
     namespace: echoserver
   spec:
     ports:
       - port: 80
         targetPort: 8080
         protocol: TCP
     type: NodePort
     selector:
       app: echoserver
   ```

   1   Defines the service resource.

   2   Specifies the service name.

5. Create a YAML file that defines the **Ingress** resource:

   Example **ingress-albo.yaml** file:

   ```yaml
   apiVersion: networking.k8s.io/v1
   kind: Ingress
   metadata:
     name: <name> 1
     namespace: echoserver
     annotations:
       alb.ingress.kubernetes.io/scheme: internet-facing
       alb.ingress.kubernetes.io/target-type: instance
   spec:
     ingressClassName: alb
     rules:
       - http:
           paths:
             - path: /
               pathType: Exact
               backend:
                 service:
                   name: <echoserver> 2
                   port:
                     number: 80
   ```

   1   Specify a name for the **Ingress** resource.

2 Specifies the service name.

Verification

- Save the status of the **Ingress** resource in the **HOST** variable by running the following command:

```
$ HOST=$(oc get ingress -n echoserver echoserver --template='{{(index
.status.loadBalancer.ingress 0).hostname}}')
```

- Verify the status of the **Ingress** resource by running the following command:

```
$ curl $HOST
```

24.6. SERVING MULTIPLE INGRESS RESOURCES THROUGH A SINGLE AWS LOAD BALANCER

You can route the traffic to different services that are part of a single domain through a single AWS Load Balancer. Each Ingress resource provides different endpoints of the domain.

24.6.1. Creating multiple ingress resources through a single AWS Load Balancer

You can route the traffic to multiple ingress resources through a single AWS Load Balancer by using the CLI.

Prerequisites

- You have an access to the OpenShift CLI (**oc**).

Procedure

1. Create an **IngressClassParams** resource YAML file, for example, **sample-single-lb-params.yaml**, as follows:

```
apiVersion: elbv2.k8s.aws/v1beta1 1
kind: IngressClassParams
metadata:
  name: single-lb-params 2
spec:
  group:
    name: single-lb 3
```

1 Defines the API group and version of the **IngressClassParams** resource.

2 Specifies the **IngressClassParams** resource name.

3 Specifies the **IngressGroup** resource name. All of the **Ingress** resources of this class belong to this **IngressGroup**.

2. Create the **IngressClassParams** resource by running the following command:

```
$ oc create -f sample-single-lb-params.yaml
```

3. Create the **IngressClass** resource YAML file, for example, **sample-single-lb-class.yaml**, as follows:

```
apiVersion: networking.k8s.io/v1 1
kind: IngressClass
metadata:
  name: single-lb 2
spec:
  controller: ingress.k8s.aws/alb 3
  parameters:
    apiGroup: elbv2.k8s.aws 4
    kind: IngressClassParams 5
    name: single-lb-params 6
```

1   Defines the API group and version of the **IngressClass** resource.

2   Specifies the ingress class name.

3   Defines the controller name. The **ingress.k8s.aws/alb** value denotes that all ingress resources of this class should be managed by the AWS Load Balancer Controller.

4   Defines the API group of the **IngressClassParams** resource.

5   Defines the resource type of the **IngressClassParams** resource.

6   Defines the **IngressClassParams** resource name.

4. Create the **IngressClass** resource by running the following command:

```
$ oc create -f sample-single-lb-class.yaml
```

5. Create the **AWSLoadBalancerController** resource YAML file, for example, **sample-single-lb.yaml**, as follows:

```
apiVersion: networking.olm.openshift.io/v1
kind: AWSLoadBalancerController
metadata:
  name: cluster
spec:
  subnetTagging: Auto
  ingressClass: single-lb 1
```

1   Defines the name of the **IngressClass** resource.

6. Create the **AWSLoadBalancerController** resource by running the following command:

```
$ oc create -f sample-single-lb.yaml
```

7. Create the **Ingress** resource YAML file, for example, **sample-multiple-ingress.yaml**, as follows:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: example-1 1
  annotations:
    alb.ingress.kubernetes.io/scheme: internet-facing 2
    alb.ingress.kubernetes.io/group.order: "1" 3
    alb.ingress.kubernetes.io/target-type: instance 4
spec:
  ingressClassName: single-lb 5
  rules:
  - host: example.com 6
    http:
      paths:
      - path: /blog 7
        pathType: Prefix
        backend:
          service:
            name: example-1 8
            port:
              number: 80 9
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: example-2
  annotations:
    alb.ingress.kubernetes.io/scheme: internet-facing
    alb.ingress.kubernetes.io/group.order: "2"
    alb.ingress.kubernetes.io/target-type: instance
spec:
  ingressClassName: single-lb
  rules:
  - host: example.com
    http:
      paths:
      - path: /store
        pathType: Prefix
        backend:
          service:
            name: example-2
            port:
              number: 80
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: example-3
  annotations:
    alb.ingress.kubernetes.io/scheme: internet-facing
    alb.ingress.kubernetes.io/group.order: "3"
    alb.ingress.kubernetes.io/target-type: instance
spec:
  ingressClassName: single-lb
  rules:
```

```
    - host: example.com
     http:
        paths:
        - path: /
          pathType: Prefix
          backend:
            service:
              name: example-3
              port:
                number: 80
```

1   Specifies the ingress name.

2   Indicates the load balancer to provision in the public subnet to access the internet.

3   Specifies the order in which the rules from the multiple ingress resources are matched when the request is received at the load balancer.

4   Indicates that the load balancer will target OpenShift Container Platform nodes to reach the service.

5   Specifies the ingress class that belongs to this ingress.

6   Defines a domain name used for request routing.

7   Defines the path that must route to the service.

8   Defines the service name that serves the endpoint configured in the **Ingress** resource.

9   Defines the port on the service that serves the endpoint.

8. Create the **Ingress** resource by running the following command:

```
$ oc create -f sample-multiple-ingress.yaml
```

24.7. ADDING TLS TERMINATION

You can add TLS termination on the AWS Load Balancer.

24.7.1. Adding TLS termination on the AWS Load Balancer

You can route the traffic for the domain to pods of a service and add TLS termination on the AWS Load Balancer.

Prerequisites

- You have an access to the OpenShift CLI (**oc**).

Procedure

1. Create a YAML file that defines the **AWSLoadBalancerController** resource:

   Example **add-tls-termination-albc.yaml** file

```
apiVersion: networking.olm.openshift.io/v1
kind: AWSLoadBalancerController
metadata:
  name: cluster
spec:
  subnetTagging: Auto
  ingressClass: tls-termination 1
```

1 Defines the ingress class name. If the ingress class is not present in your cluster the AWS Load Balancer Controller creates one. The AWS Load Balancer Controller reconciles the additional ingress class values if **spec.controller** is set to **ingress.k8s.aws/alb**.

2. Create a YAML file that defines the **Ingress** resource:

   Example **add-tls-termination-ingress.yaml** file

   ```
   apiVersion: networking.k8s.io/v1
   kind: Ingress
   metadata:
     name: <example> 1
     annotations:
       alb.ingress.kubernetes.io/scheme: internet-facing 2
       alb.ingress.kubernetes.io/certificate-arn: arn:aws:acm:us-west-2:xxxxx 3
   spec:
     ingressClassName: tls-termination 4
     rules:
     - host: <example.com> 5
       http:
           paths:
             - path: /
               pathType: Exact
               backend:
                 service:
                   name: <example-service> 6
                   port:
                     number: 80
   ```

1 Specifies the ingress name.

2 The controller provisions the load balancer for ingress in a public subnet to access the load balancer over the internet.

3 The Amazon Resource Name (ARN) of the certificate that you attach to the load balancer.

4 Defines the ingress class name.

5 Defines the domain for traffic routing.

6 Defines the service for traffic routing.

24.8. CONFIGURING CLUSTER-WIDE PROXY

You can configure the cluster-wide proxy in the AWS Load Balancer Operator. After configuring the cluster-wide proxy, Operator Lifecycle Manager (OLM) automatically updates all the deployments of the Operators with the environment variables such as **HTTP_PROXY**, **HTTPS_PROXY**, and **NO_PROXY**. These variables are populated to the managed controller by the AWS Load Balancer Operator.

24.8.1. Trusting the certificate authority of the cluster-wide proxy

1. Create the config map to contain the certificate authority (CA) bundle in the **aws-load-balancer-operator** namespace by running the following command:

   ```
   $ oc -n aws-load-balancer-operator create configmap trusted-ca
   ```

2. To inject the trusted CA bundle into the config map, add the **config.openshift.io/inject-trusted-cabundle=true** label to the config map by running the following command:

   ```
   $ oc -n aws-load-balancer-operator label cm trusted-ca config.openshift.io/inject-trusted-cabundle=true
   ```

3. Update the AWS Load Balancer Operator subscription to access the config map in the AWS Load Balancer Operator deployment by running the following command:

   ```
   $ oc -n aws-load-balancer-operator patch subscription aws-load-balancer-operator --type='merge' -p '{"spec":{"config":{"env":
   [{"name":"TRUSTED_CA_CONFIGMAP_NAME","value":"trusted-ca"}],"volumes":
   [{"name":"trusted-ca","configMap":{"name":"trusted-ca"}}],"volumeMounts":[{"name":"trusted-ca","mountPath":"/etc/pki/tls/certs/albo-tls-ca-bundle.crt","subPath":"ca-bundle.crt"}]}}}'
   ```

4. After the AWS Load Balancer Operator is deployed, verify that the CA bundle is added to the **aws-load-balancer-operator-controller-manager** deployment by running the following command:

   ```
   $ oc -n aws-load-balancer-operator exec deploy/aws-load-balancer-operator-controller-manager -c manager -- bash -c "ls -l /etc/pki/tls/certs/albo-tls-ca-bundle.crt; printenv TRUSTED_CA_CONFIGMAP_NAME"
   ```

   Example output

   ```
   -rw-r--r--. 1 root 1000690000 5875 Jan 11 12:25 /etc/pki/tls/certs/albo-tls-ca-bundle.crt
   trusted-ca
   ```

5. Optional: Restart deployment of the AWS Load Balancer Operator every time the config map changes by running the following command:

   ```
   $ oc -n aws-load-balancer-operator rollout restart deployment/aws-load-balancer-operator-controller-manager
   ```

24.8.2. Additional resources

- [Certificate injection using Operators](#)

CHAPTER 25. MULTIPLE NETWORKS

25.1. UNDERSTANDING MULTIPLE NETWORKS

In Kubernetes, container networking is delegated to networking plugins that implement the Container Network Interface (CNI).

OpenShift Container Platform uses the Multus CNI plugin to allow chaining of CNI plugins. During cluster installation, you configure your *default* pod network. The default network handles all ordinary network traffic for the cluster. You can define an *additional network* based on the available CNI plugins and attach one or more of these networks to your pods. You can define more than one additional network for your cluster, depending on your needs. This gives you flexibility when you configure pods that deliver network functionality, such as switching or routing.

25.1.1. Usage scenarios for an additional network

You can use an additional network in situations where network isolation is needed, including data plane and control plane separation. Isolating network traffic is useful for the following performance and security reasons:

Performance

> You can send traffic on two different planes to manage how much traffic is along each plane.

Security

> You can send sensitive traffic onto a network plane that is managed specifically for security considerations, and you can separate private data that must not be shared between tenants or customers.

All of the pods in the cluster still use the cluster-wide default network to maintain connectivity across the cluster. Every pod has an **eth0** interface that is attached to the cluster-wide pod network. You can view the interfaces for a pod by using the **oc exec -it <pod_name> -- ip a** command. If you add additional network interfaces that use Multus CNI, they are named **net1**, **net2**, …, **netN**.

To attach additional network interfaces to a pod, you must create configurations that define how the interfaces are attached. You specify each interface by using a **NetworkAttachmentDefinition** custom resource (CR). A CNI configuration inside each of these CRs defines how that interface is created.

25.1.2. Additional networks in OpenShift Container Platform

OpenShift Container Platform provides the following CNI plugins for creating additional networks in your cluster:

- **bridge**: Configure a bridge-based additional network to allow pods on the same host to communicate with each other and the host.

- **host-device**: Configure a host-device additional network to allow pods access to a physical Ethernet network device on the host system.

- **ipvlan**: Configure an ipvlan-based additional network to allow pods on a host to communicate with other hosts and pods on those hosts, similar to a macvlan-based additional network. Unlike a macvlan-based additional network, each pod shares the same MAC address as the parent physical network interface.

- **vlan**: Configure a vlan-based additional network to allow VLAN-based network isolation and connectivity for pods.

- **macvlan**: Configure a macvlan-based additional network to allow pods on a host to communicate with other hosts and pods on those hosts by using a physical network interface. Each pod that is attached to a macvlan-based additional network is provided a unique MAC address.

- **SR-IOV**: Configure an SR-IOV based additional network to allow pods to attach to a virtual function (VF) interface on SR-IOV capable hardware on the host system.

25.2. CONFIGURING AN ADDITIONAL NETWORK

As a cluster administrator, you can configure an additional network for your cluster. The following network types are supported:

- Bridge

- Host device

- VLAN

- IPVLAN

- MACVLAN

- OVN-Kubernetes

25.2.1. Approaches to managing an additional network

You can manage the life cycle of an additional network by two approaches. Each approach is mutually exclusive and you can only use one approach for managing an additional network at a time. For either approach, the additional network is managed by a Container Network Interface (CNI) plugin that you configure.

For an additional network, IP addresses are provisioned through an IP Address Management (IPAM) CNI plugin that you configure as part of the additional network. The IPAM plugin supports a variety of IP address assignment approaches including DHCP and static assignment.

- Modify the Cluster Network Operator (CNO) configuration: The CNO automatically creates and manages the **NetworkAttachmentDefinition** object. In addition to managing the object lifecycle the CNO ensures a DHCP is available for an additional network that uses a DHCP assigned IP address.

- Applying a YAML manifest: You can manage the additional network directly by creating an **NetworkAttachmentDefinition** object. This approach allows for the chaining of CNI plugins.

> **NOTE**
>
> When deploying OpenShift Container Platform nodes with multiple network interfaces on Red Hat OpenStack Platform (RHOSP) with OVN SDN, DNS configuration of the secondary interface might take precedence over the DNS configuration of the primary interface. In this case, remove the DNS nameservers for the subnet id that is attached to the secondary interface:
>
> ```
> $ openstack subnet set --dns-nameserver 0.0.0.0 <subnet_id>
> ```

25.2.2. Configuration for an additional network attachment

An additional network is configured by using the **NetworkAttachmentDefinition** API in the **k8s.cni.cncf.io** API group.
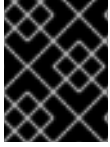
> **IMPORTANT**
>
> Do not store any sensitive information or a secret in the **NetworkAttachmentDefinition** object because this information is accessible by the project administration user.

The configuration for the API is described in the following table:

Table 25.1. **NetworkAttachmentDefinition** API fields

Field	Type	Description
metadata.name	**string**	The name for the additional network.
metadata.namespace	**string**	The namespace that the object is associated with.
spec.config	**string**	The CNI plugin configuration in JSON format.

25.2.2.1. Configuration of an additional network through the Cluster Network Operator

The configuration for an additional network attachment is specified as part of the Cluster Network Operator (CNO) configuration.

The following YAML describes the configuration parameters for managing an additional network with the CNO:

Cluster Network Operator configuration

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  # ...
  additionalNetworks: 1
  - name: <name> 2
    namespace: <namespace> 3
    rawCNIConfig: |- 4
      {
        ...
      }
    type: Raw
```

1 An array of one or more additional network configurations.

2 The name for the additional network attachment that you are creating. The name must be unique within the specified **namespace**.

3 The namespace to create the network attachment in. If you do not specify a value, then the **default** namespace is used.

4 A CNI plugin configuration in JSON format.

25.2.2.2. Configuration of an additional network from a YAML manifest

The configuration for an additional network is specified from a YAML configuration file, such as in the following example:

```
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  name: <name> 1
spec:
  config: |- 2
    {
      ...
    }
```

1 The name for the additional network attachment that you are creating.

2 A CNI plugin configuration in JSON format.

25.2.3. Configurations for additional network types

The specific configuration fields for additional networks is described in the following sections.

25.2.3.1. Configuration for a bridge additional network

The following object describes the configuration parameters for the bridge CNI plugin:

Table 25.2. Bridge CNI plugin JSON configuration object

Field	Type	Description
cniVersion	**string**	The CNI specification version. The **0.3.1** value is required.
name	**string**	The value for the **name** parameter you provided previously for the CNO configuration.
type	**string**	The name of the CNI plugin to configure: **bridge**.
ipam	**object**	The configuration object for the IPAM CNI plugin. The plugin manages IP address assignment for the attachment definition.
bridge	**string**	Optional: Specify the name of the virtual bridge to use. If the bridge interface does not exist on the host, it is created. The default value is **cni0**.

Field	Type	Description
ipMasq	**boolean**	Optional: Set to **true** to enable IP masquerading for traffic that leaves the virtual network. The source IP address for all traffic is rewritten to the bridge's IP address. If the bridge does not have an IP address, this setting has no effect. The default value is **false**.
isGateway	**boolean**	Optional: Set to **true** to assign an IP address to the bridge. The default value is **false**.
isDefaultGateway	**boolean**	Optional: Set to **true** to configure the bridge as the default gateway for the virtual network. The default value is **false**. If **isDefaultGateway** is set to **true**, then **isGateway** is also set to **true** automatically.
forceAddress	**boolean**	Optional: Set to **true** to allow assignment of a previously assigned IP address to the virtual bridge. When set to **false**, if an IPv4 address or an IPv6 address from overlapping subsets is assigned to the virtual bridge, an error occurs. The default value is **false**.
hairpinMode	**boolean**	Optional: Set to **true** to allow the virtual bridge to send an Ethernet frame back through the virtual port it was received on. This mode is also known as *reflective relay*. The default value is **false**.
promiscMode	**boolean**	Optional: Set to **true** to enable promiscuous mode on the bridge. The default value is **false**.
vlan	**string**	Optional: Specify a virtual LAN (VLAN) tag as an integer value. By default, no VLAN tag is assigned.
preserveDefaultVlan	**string**	Optional: Indicates whether the default vlan must be preserved on the **veth** end connected to the bridge. Defaults to true.
vlanTrunk	**list**	Optional: Assign a VLAN trunk tag. The default value is **none**.
mtu	**string**	Optional: Set the maximum transmission unit (MTU) to the specified value. The default value is automatically set by the kernel.
enabledad	**boolean**	Optional: Enables duplicate address detection for the container side **veth**. The default value is **false**.
macspoofchk	**boolean**	Optional: Enables mac spoof check, limiting the traffic originating from the container to the mac address of the interface. The default value is **false**.

> **NOTE**
>
> The VLAN parameter configures the VLAN tag on the host end of the **veth** and also enables the **vlan_filtering** feature on the bridge interface.

> **NOTE**
>
> To configure uplink for a L2 network you need to allow the vlan on the uplink interface by using the following command:
>
> ```
> $ bridge vlan add vid VLAN_ID dev DEV
> ```

25.2.3.1.1. bridge configuration example

The following example configures an additional network named **bridge-net**:

```
{
  "cniVersion": "0.3.1",
  "name": "bridge-net",
  "type": "bridge",
  "isGateway": true,
  "vlan": 2,
  "ipam": {
    "type": "dhcp"
    }
}
```

25.2.3.2. Configuration for a host device additional network

> **NOTE**
>
> Specify your network device by setting only one of the following parameters: **device**,**hwaddr**, **kernelpath**, or **pciBusID**.

The following object describes the configuration parameters for the host-device CNI plugin:

Table 25.3. Host device CNI plugin JSON configuration object

Field	Type	Description
cniVersion	**string**	The CNI specification version. The **0.3.1** value is required.
name	**string**	The value for the **name** parameter you provided previously for the CNO configuration.
type	**string**	The name of the CNI plugin to configure: **host-device**.
device	**string**	Optional: The name of the device, such as **eth0**.
hwaddr	**string**	Optional: The device hardware MAC address.

Field	Type	Description
kernelpath	**string**	Optional: The Linux kernel device path, such as **/sys/devices/pci0000:00/0000:00:1f.6**.
pciBusID	**string**	Optional: The PCI address of the network device, such as **0000:00:1f.6**.

25.2.3.2.1. host-device configuration example

The following example configures an additional network named **hostdev-net**:

```
{
  "cniVersion": "0.3.1",
  "name": "hostdev-net",
  "type": "host-device",
  "device": "eth1"
}
```

25.2.3.3. Configuration for an VLAN additional network

The following object describes the configuration parameters for the VLAN CNI plugin:

Table 25.4. VLAN CNI plugin JSON configuration object

Field	Type	Description
cniVersion	**string**	The CNI specification version. The **0.3.1** value is required.
name	**string**	The value for the **name** parameter you provided previously for the CNO configuration.
type	**string**	The name of the CNI plugin to configure: **vlan**.
master	**string**	The Ethernet interface to associate with the network attachment. If a **master** is not specified, the interface for the default network route is used.
vlanId	**integer**	Set the id of the vlan.
ipam	**object**	The configuration object for the IPAM CNI plugin. The plugin manages IP address assignment for the attachment definition.
mtu	**integer**	Optional: Set the maximum transmission unit (MTU) to the specified value. The default value is automatically set by the kernel.
dns	**integer**	Optional: DNS information to return, for example, a priority-ordered list of DNS nameservers.

Field	Type	Description
linkInContainer	**boolean**	Optional: Specifies if the master interface is in the container network namespace or the main network namespace.

25.2.3.3.1. vlan configuration example

The following example configures an additional network named **vlan-net**:

```
{
  "name": "vlan-net",
  "cniVersion": "0.3.1",
  "type": "vlan",
  "master": "eth0",
  "mtu": 1500,
  "vlanId": 5,
  "linkInContainer": false,
  "ipam": {
      "type": "host-local",
      "subnet": "10.1.1.0/24"
  },
  "dns": {
      "nameservers": [ "10.1.1.1", "8.8.8.8" ]
  }
}
```

25.2.3.4. Configuration for an IPVLAN additional network

The following object describes the configuration parameters for the IPVLAN CNI plugin:

Table 25.5. IPVLAN CNI plugin JSON configuration object

Field	Type	Description
cniVersion	**string**	The CNI specification version. The **0.3.1** value is required.
name	**string**	The value for the **name** parameter you provided previously for the CNO configuration.
type	**string**	The name of the CNI plugin to configure: **ipvlan**.
ipam	**object**	The configuration object for the IPAM CNI plugin. The plugin manages IP address assignment for the attachment definition. This is required unless the plugin is chained.
mode	**string**	Optional: The operating mode for the virtual network. The value must be **l2**, **l3**, or **l3s**. The default value is **l2**.

Field	Type	Description
master	**string**	Optional: The Ethernet interface to associate with the network attachment. If a **master** is not specified, the interface for the default network route is used.
mtu	**integer**	Optional: Set the maximum transmission unit (MTU) to the specified value. The default value is automatically set by the kernel.

NOTE

- The **ipvlan** object does not allow virtual interfaces to communicate with the **master** interface. Therefore the container will not be able to reach the host by using the **ipvlan** interface. Be sure that the container joins a network that provides connectivity to the host, such as a network supporting the Precision Time Protocol (**PTP**).

- A single **master** interface cannot simultaneously be configured to use both **macvlan** and **ipvlan**.

- For IP allocation schemes that cannot be interface agnostic, the **ipvlan** plugin can be chained with an earlier plugin that handles this logic. If the **master** is omitted, then the previous result must contain a single interface name for the **ipvlan** plugin to enslave. If **ipam** is omitted, then the previous result is used to configure the **ipvlan** interface.

25.2.3.4.1. ipvlan configuration example

The following example configures an additional network named **ipvlan-net**:

```
{
  "cniVersion": "0.3.1",
  "name": "ipvlan-net",
  "type": "ipvlan",
  "master": "eth1",
  "mode": "l3",
  "ipam": {
    "type": "static",
    "addresses": [
      {
        "address": "192.168.10.10/24"
      }
    ]
  }
}
```

25.2.3.5. Configuration for a MACVLAN additional network

The following object describes the configuration parameters for the macvlan CNI plugin:

Table 25.6. MACVLAN CNI plugin JSON configuration object

Field	Type	Description
cniVersion	**string**	The CNI specification version. The **0.3.1** value is required.
name	**string**	The value for the **name** parameter you provided previously for the CNO configuration.
type	**string**	The name of the CNI plugin to configure: **macvlan**.
ipam	**object**	The configuration object for the IPAM CNI plugin. The plugin manages IP address assignment for the attachment definition.
mode	**string**	Optional: Configures traffic visibility on the virtual network. Must be either **bridge**, **passthru**, **private**, or **vepa**. If a value is not provided, the default value is **bridge**.
master	**string**	Optional: The host network interface to associate with the newly created macvlan interface. If a value is not specified, then the default route interface is used.
mtu	**string**	Optional: The maximum transmission unit (MTU) to the specified value. The default value is automatically set by the kernel.

> **NOTE**
>
> If you specify the **master** key for the plugin configuration, use a different physical network interface than the one that is associated with your primary network plugin to avoid possible conflicts.

25.2.3.5.1. macvlan configuration example

The following example configures an additional network named **macvlan-net**:

```
{
  "cniVersion": "0.3.1",
  "name": "macvlan-net",
  "type": "macvlan",
  "master": "eth1",
  "mode": "bridge",
  "ipam": {
    "type": "dhcp"
    }
}
```

25.2.3.6. Configuration for an OVN-Kubernetes additional network

The Red Hat OpenShift Networking OVN-Kubernetes network plugin allows the configuration of secondary network interfaces for pods. To configure secondary network interfaces, you must define the configurations in the **NetworkAttachmentDefinition** custom resource definition (CRD).

IMPORTANT

Configuration for an OVN-Kubernetes additional network is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

NOTE

Pod and multi-network policy creation might remain in a pending state until the OVN-Kubernetes control plane agent in the nodes processes the associated **network-attachment-definition** CR.

The following sections provide example configurations for each of the topologies that OVN-Kubernetes currently allows for secondary networks.

NOTE

Networks names must be unique. For example, creating multiple **NetworkAttachmentDefinition** CRDs with different configurations that reference the same network is unsupported.

25.2.3.6.1. OVN-Kubernetes network plugin JSON configuration table

The following table describes the configuration parameters for the OVN-Kubernetes CNI network plugin:

Table 25.7. OVN-Kubernetes network plugin JSON configuration table

Field	Type	Description
cniVersion	**string**	The CNI specification version. The required value is **0.3.1**.
name	**string**	The name of the network. These networks are not namespaced. For example, you can have a network named **l2-network** referenced from two different **NetworkAttachmentDefinitions** that exist on two different namespaces. This ensures that pods making use of the **NetworkAttachmentDefinition** on their own different namespaces can communicate over the same secondary network. However, those two different **NetworkAttachmentDefinitions** must also share the same network specific parameters such as **topology**, **subnets**, **mtu**, and **excludeSubnets**.
type	**string**	The name of the CNI plugin to configure. The required value is **ovn-k8s-cni-overlay**.

Field	Type	Description
topology	**string**	The topological configuration for the network. The required value is **layer2**.
subnets	**string**	The subnet to use for the network across the cluster. When specifying **layer2** for the **topology**, only include the CIDR for the node. For example, **10.100.200.0/24**. For **"topology":"layer2"** deployments, IPv6 (**2001:DBB::/64**) and dual-stack (**192.168.100.0/24,2001:DBB::/64**) subnets are supported.
mtu	**string**	The maximum transmission unit (MTU) to the specified value. The default value, **1300**, is automatically set by the kernel.
netAttachDefName me	**string**	The metadata **namespace** and **name** of the network attachment definition object where this configuration is included. For example, if this configuration is defined in a **NetworkAttachmentDefinition** in namespace **ns1** named **l2-network**, this should be set to **ns1/l2-network**.
excludeSubnets	**string**	A comma-separated list of CIDRs and IPs. IPs are removed from the assignable IP pool, and are never passed to the pods. When omitted, the logical switch implementing the network only provides layer 2 communication, and users must configure IPs for the pods. Port security only prevents MAC spoofing.

25.2.3.6.2. Configuration for a switched topology

The switched (layer 2) topology networks interconnect the workloads through a cluster-wide logical switch. This configuration can be used for IPv6 and dual-stack deployments.

> **NOTE**
>
> Layer 2 switched topology networks only allow for the transfer of data packets between pods within a cluster.

The following **NetworkAttachmentDefinition** custom resource definition (CRD) YAML describes the fields needed to configure a switched secondary network.

```
{
    "cniVersion": "0.3.1",
    "name": "l2-network",
    "type": "ovn-k8s-cni-overlay",
    "topology":"layer2",
    "subnets": "10.100.200.0/24",
    "mtu": 1300,
    "netAttachDefName": "ns1/l2-network",
    "excludeSubnets": "10.100.200.0/29"
}
```

25.2.3.6.3. Configuring pods for additional networks

You must specify the secondary network attachments through the **k8s.v1.cni.cncf.io/networks** annotation.

The following example provisions a pod with two secondary attachments, one for each of the attachment configurations presented in this guide.

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: l2-network
  name: tinypod
  namespace: ns1
spec:
  containers:
  - args:
    - pause
    image: k8s.gcr.io/e2e-test-images/agnhost:2.36
    imagePullPolicy: IfNotPresent
    name: agnhost-container
```

25.2.3.6.4. Configuring pods with a static IP address

The following example provisions a pod with a static IP address.
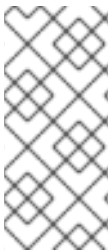
> **NOTE**
>
> - You can only specify the IP address for a pod's secondary network attachment for layer 2 attachments.
>
> - Specifying a static IP address for the pod is only possible when the attachment configuration does not feature subnets.

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: '[
      {
        "name": "l2-network",          1
        "mac": "02:03:04:05:06:07",    2
        "interface": "myiface1",       3
        "ips": [
          "192.0.2.20/24"
        ]                              4
      }
    ]'
  name: tinypod
  namespace: ns1
spec:
  containers:
  - args:
```

```
- pause
  image: k8s.gcr.io/e2e-test-images/agnhost:2.36
  imagePullPolicy: IfNotPresent
  name: agnhost-container
```

1   The name of the network. This value must be unique across all **NetworkAttachmentDefinitions**.

2   The MAC address to be assigned for the interface.

3   The name of the network interface to be created for the pod.

4   The IP addresses to be assigned to the network interface.

25.2.4. Configuration of IP address assignment for an additional network

The IP address management (IPAM) Container Network Interface (CNI) plugin provides IP addresses for other CNI plugins.

You can use the following IP address assignment types:

- Static assignment.

- Dynamic assignment through a DHCP server. The DHCP server you specify must be reachable from the additional network.

- Dynamic assignment through the Whereabouts IPAM CNI plugin.

25.2.4.1. Static IP address assignment configuration

The following table describes the configuration for static IP address assignment:

Table 25.8. **ipam** static configuration object

Field	Type	Description
type	**string**	The IPAM address type. The value **static** is required.
addresses	**array**	An array of objects specifying IP addresses to assign to the virtual interface. Both IPv4 and IPv6 IP addresses are supported.
routes	**array**	An array of objects specifying routes to configure inside the pod.
dns	**array**	Optional: An array of objects specifying the DNS configuration.

The **addresses** array requires objects with the following fields:

Table 25.9. **ipam.addresses[]** array

Field	Type	Description

Field	Type	Description
address	**string**	An IP address and network prefix that you specify. For example, if you specify **10.10.21.10/24**, then the additional network is assigned an IP address of **10.10.21.10** and the netmask is **255.255.255.0**.
gateway	**string**	The default gateway to route egress network traffic to.

Table 25.10. **ipam.routes[]** array

Field	Type	Description
dst	**string**	The IP address range in CIDR format, such as **192.168.17.0/24** or **0.0.0.0/0** for the default route.
gw	**string**	The gateway where network traffic is routed.

Table 25.11. **ipam.dns** object

Field	Type	Description
nameservers	**array**	An array of one or more IP addresses for to send DNS queries to.
domain	**array**	The default domain to append to a hostname. For example, if the domain is set to **example.com**, a DNS lookup query for **example-host** is rewritten as **example-host.example.com**.
search	**array**	An array of domain names to append to an unqualified hostname, such as **example-host**, during a DNS lookup query.

Static IP address assignment configuration example

```
{
  "ipam": {
    "type": "static",
      "addresses": [
        {
          "address": "191.168.1.7/24"
        }
      ]
  }
}
```

25.2.4.2. Dynamic IP address (DHCP) assignment configuration

The following JSON describes the configuration for dynamic IP address address assignment with DHCP.

RENEWAL OF DHCP LEASES

A pod obtains its original DHCP lease when it is created. The lease must be periodically renewed by a minimal DHCP server deployment running on the cluster.

To trigger the deployment of the DHCP server, you must create a shim network attachment by editing the Cluster Network Operator configuration, as in the following example:

Example shim network attachment definition

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  additionalNetworks:
  - name: dhcp-shim
    namespace: default
    type: Raw
    rawCNIConfig: |-
      {
        "name": "dhcp-shim",
        "cniVersion": "0.3.1",
        "type": "bridge",
        "ipam": {
          "type": "dhcp"
        }
      }
  # ...
```

Table 25.12. ipam DHCP configuration object

Field	Type	Description
type	**string**	The IPAM address type. The value **dhcp** is required.

Dynamic IP address (DHCP) assignment configuration example

```
{
  "ipam": {
    "type": "dhcp"
  }
}
```

25.2.4.3. Dynamic IP address assignment configuration with Whereabouts

The Whereabouts CNI plugin allows the dynamic assignment of an IP address to an additional network without the use of a DHCP server.

The following table describes the configuration for dynamic IP address assignment with Whereabouts:

Table 25.13. ipam whereabouts configuration object

Field	Type	Description
type	**string**	The IPAM address type. The value **whereabouts** is required.
range	**string**	An IP address and range in CIDR notation. IP addresses are assigned from within this range of addresses.
exclude	**array**	Optional: A list of zero or more IP addresses and ranges in CIDR notation. IP addresses within an excluded address range are not assigned.

Dynamic IP address assignment configuration example that uses Whereabouts

```
{
  "ipam": {
    "type": "whereabouts",
    "range": "192.0.2.192/27",
    "exclude": [
       "192.0.2.192/30",
       "192.0.2.196/32"
    ]
  }
}
```

25.2.4.4. Creating a whereabouts-reconciler daemon set

The Whereabouts reconciler is responsible for managing dynamic IP address assignments for the pods within a cluster by using the Whereabouts IP Address Management (IPAM) solution. It ensures that each pod gets a unique IP address from the specified IP address range. It also handles IP address releases when pods are deleted or scaled down.

> **NOTE**
>
> You can also use a **NetworkAttachmentDefinition** custom resource (CR) for dynamic IP address assignment.

The **whereabouts-reconciler** daemon set is automatically created when you configure an additional network through the Cluster Network Operator. It is not automatically created when you configure an additional network from a YAML manifest.

To trigger the deployment of the **whereabouts-reconciler** daemon set, you must manually create a **whereabouts-shim** network attachment by editing the Cluster Network Operator custom resource (CR) file.

Use the following procedure to deploy the **whereabouts-reconciler** daemon set.

Procedure

1. Edit the **Network.operator.openshift.io** custom resource (CR) by running the following command:

```
$ oc edit network.operator.openshift.io cluster
```

2. Include the **additionalNetworks** section shown in this example YAML extract within the **spec** definition of the custom resource (CR):

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
# ...
spec:
  additionalNetworks:
  - name: whereabouts-shim
    namespace: default
    rawCNIConfig: |-
      {
       "name": "whereabouts-shim",
       "cniVersion": "0.3.1",
       "type": "bridge",
       "ipam": {
         "type": "whereabouts"
       }
      }
    type: Raw
# ...
```

3. Save the file and exit the text editor.

4. Verify that the **whereabouts-reconciler** daemon set deployed successfully by running the following command:

```
$ oc get all -n openshift-multus | grep whereabouts-reconciler
```

Example output

```
pod/whereabouts-reconciler-jnp6g 1/1 Running 0 6s
pod/whereabouts-reconciler-k76gg 1/1 Running 0 6s
pod/whereabouts-reconciler-k86t9 1/1 Running 0 6s
pod/whereabouts-reconciler-p4sxw 1/1 Running 0 6s
pod/whereabouts-reconciler-rvfdv 1/1 Running 0 6s
pod/whereabouts-reconciler-svzw9 1/1 Running 0 6s
daemonset.apps/whereabouts-reconciler 6 6 6 6 6 kubernetes.io/os=linux 6s
```

25.2.4.5. Configuring the Whereabouts IP reconciler schedule

The Whereabouts IPAM CNI plugin runs the IP reconciler daily. This process cleans up any stranded IP allocations that might result in exhausting IPs and therefore prevent new pods from getting an IP allocated to them.

Use this procedure to change the frequency at which the IP reconciler runs.

Prerequisites

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

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

- You have deployed the **whereabouts-reconciler** daemon set, and the **whereabouts-reconciler** pods are up and running.

Procedure

1. Run the following command to create a **ConfigMap** object named **whereabouts-config** in the **openshift-multus** namespace with a specific cron expression for the IP reconciler:

   ```
   $ oc create configmap whereabouts-config -n openshift-multus --from-
   literal=reconciler_cron_expression="*/15 * * * *"
   ```

   This cron expression indicates the IP reconciler runs every 15 minutes. Adjust the expression based on your specific requirements.

   > **NOTE**
   >
   > The **whereabouts-reconciler** daemon set can only consume a cron expression pattern that includes five asterisks. The sixth, which is used to denote seconds, is currently not supported.

2. Retrieve information about resources related to the **whereabouts-reconciler** daemon set and pods within the **openshift-multus** namespace by running the following command:

   ```
   $ oc get all -n openshift-multus | grep whereabouts-reconciler
   ```

   Example output

   ```
   pod/whereabouts-reconciler-2p7hw                 1/1     Running   0         4m14s
   pod/whereabouts-reconciler-76jk7                 1/1     Running   0         4m14s
   pod/whereabouts-reconciler-94zw6                 1/1     Running   0         4m14s
   pod/whereabouts-reconciler-mfh68                 1/1     Running   0         4m14s
   pod/whereabouts-reconciler-pgshz                 1/1     Running   0         4m14s
   pod/whereabouts-reconciler-xn5xz                 1/1     Running   0         4m14s
   daemonset.apps/whereabouts-reconciler            6       6       6      6        6
   kubernetes.io/os=linux   4m16s
   ```

3. Run the following command to verify that the **whereabouts-reconciler** pod runs the IP reconciler with the configured interval:

   ```
   $ oc -n openshift-multus logs whereabouts-reconciler-2p7hw
   ```

   Example output

   ```
   2024-02-02T16:33:54Z [debug] event not relevant: "/cron-
   schedule/..2024_02_02_16_33_54.1375928161": CREATE
   2024-02-02T16:33:54Z [debug] event not relevant: "/cron-
   schedule/..2024_02_02_16_33_54.1375928161": CHMOD
   2024-02-02T16:33:54Z [debug] event not relevant: "/cron-schedule/..data_tmp": RENAME
   2024-02-02T16:33:54Z [verbose] using expression: */15 * * * *
   2024-02-02T16:33:54Z [verbose] configuration updated to file "/cron-schedule/..data". New
   cron expression: */15 * * * *
   ```

```
2024-02-02T16:33:54Z [verbose] successfully updated CRON configuration id "00c2d1c9-
631d-403f-bb86-73ad104a6817" - new cron expression: */15 * * * *
2024-02-02T16:33:54Z [debug] event not relevant: "/cron-schedule/config": CREATE
2024-02-02T16:33:54Z [debug] event not relevant: "/cron-
schedule/..2024_02_02_16_26_17.3874177937": REMOVE
2024-02-02T16:45:00Z [verbose] starting reconciler run
2024-02-02T16:45:00Z [debug] NewReconcileLooper - inferred connection data
2024-02-02T16:45:00Z [debug] listing IP pools
2024-02-02T16:45:00Z [debug] no IP addresses to cleanup
2024-02-02T16:45:00Z [verbose] reconciler success
```

25.2.5. Creating an additional network attachment with the Cluster Network Operator

The Cluster Network Operator (CNO) manages additional network definitions. When you specify an additional network to create, the CNO creates the **NetworkAttachmentDefinition** object automatically.

IMPORTANT

Do not edit the **NetworkAttachmentDefinition** objects that the Cluster Network Operator manages. Doing so might disrupt network traffic on your additional network.

Prerequisites

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

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

Procedure

1. Optional: Create the namespace for the additional networks:

   ```
   $ oc create namespace <namespace_name>
   ```

2. To edit the CNO configuration, enter the following command:

   ```
   $ oc edit networks.operator.openshift.io cluster
   ```

3. Modify the CR that you are creating by adding the configuration for the additional network that you are creating, as in the following example CR.

   ```
   apiVersion: operator.openshift.io/v1
   kind: Network
   metadata:
     name: cluster
   spec:
     # ...
     additionalNetworks:
     - name: tertiary-net
       namespace: namespace2
       type: Raw
       rawCNIConfig: |-
         {
           "cniVersion": "0.3.1",
   ```

```
      "name": "tertiary-net",
      "type": "ipvlan",
      "master": "eth1",
      "mode": "l2",
      "ipam": {
        "type": "static",
        "addresses": [
          {
            "address": "192.168.1.23/24"
          }
        ]
      }
    }
```

4. Save your changes and quit the text editor to commit your changes.

Verification

- Confirm that the CNO created the **NetworkAttachmentDefinition** object by running the following command. There might be a delay before the CNO creates the object.

  ```
  $ oc get network-attachment-definitions -n <namespace>
  ```

  where:

  <namespace>

  Specifies the namespace for the network attachment that you added to the CNO configuration.

  Example output

  ```
  NAME            AGE
  test-network-1    14m
  ```

25.2.6. Creating an additional network attachment by applying a YAML manifest

Prerequisites

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

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

Procedure

1. Create a YAML file with your additional network configuration, such as in the following example:

   ```
   apiVersion: k8s.cni.cncf.io/v1
   kind: NetworkAttachmentDefinition
   metadata:
     name: next-net
   spec:
     config: |-
       {
   ```

```
      "cniVersion": "0.3.1",
      "name": "work-network",
      "type": "host-device",
      "device": "eth1",
      "ipam": {
        "type": "dhcp"
      }
    }
```

2. To create the additional network, enter the following command:

```
$ oc apply -f <file>.yaml
```

where:

<file>

Specifies the name of the file contained the YAML manifest.

25.3. ABOUT VIRTUAL ROUTING AND FORWARDING

25.3.1. About virtual routing and forwarding

Virtual routing and forwarding (VRF) devices combined with IP rules provide the ability to create virtual routing and forwarding domains. VRF reduces the number of permissions needed by CNF, and provides increased visibility of the network topology of secondary networks. VRF is used to provide multi-tenancy functionality, for example, where each tenant has its own unique routing tables and requires different default gateways.

Processes can bind a socket to the VRF device. Packets through the binded socket use the routing table associated with the VRF device. An important feature of VRF is that it impacts only OSI model layer 3 traffic and above so L2 tools, such as LLDP, are not affected. This allows higher priority IP rules such as policy based routing to take precedence over the VRF device rules directing specific traffic.

25.3.1.1. Benefits of secondary networks for pods for telecommunications operators

In telecommunications use cases, each CNF can potentially be connected to multiple different networks sharing the same address space. These secondary networks can potentially conflict with the cluster's main network CIDR. Using the CNI VRF plugin, network functions can be connected to different customers' infrastructure using the same IP address, keeping different customers isolated. IP addresses are overlapped with OpenShift Container Platform IP space. The CNI VRF plugin also reduces the number of permissions needed by CNF and increases the visibility of network topologies of secondary networks.

25.4. CONFIGURING MULTI-NETWORK POLICY

As a cluster administrator, you can configure multi-network for additional networks. You can specify multi-network policy for SR-IOV and macvlan additional networks. Macvlan additional networks are fully supported. Other types of additional networks, such as ipvlan, are not supported.
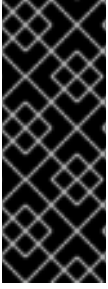
> **IMPORTANT**
>
> Support for configuring multi-network policies for SR-IOV additional networks is a Technology Preview feature and is only supported with kernel network interface cards (NICs). SR-IOV is not supported for Data Plane Development Kit (DPDK) applications.
>
> For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .
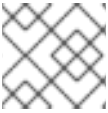
> **NOTE**
>
> Configured network policies are ignored in IPv6 networks.

25.4.1. Differences between multi-network policy and network policy

Although the **MultiNetworkPolicy** API implements the **NetworkPolicy** API, there are several important differences:

- You must use the **MultiNetworkPolicy** API:

  ```
  apiVersion: k8s.cni.cncf.io/v1beta1
  kind: MultiNetworkPolicy
  ```

- You must use the **multi-networkpolicy** resource name when using the CLI to interact with multi-network policies. For example, you can view a multi-network policy object with the **oc get multi-networkpolicy <name>** command where **<name>** is the name of a multi-network policy.

- You must specify an annotation with the name of the network attachment definition that defines the macvlan or SR-IOV additional network:

  ```
  apiVersion: k8s.cni.cncf.io/v1beta1
  kind: MultiNetworkPolicy
  metadata:
    annotations:
      k8s.v1.cni.cncf.io/policy-for: <network_name>
  ```

  where:

  <network_name>

    Specifies the name of a network attachment definition.

25.4.2. Enabling multi-network policy for the cluster

As a cluster administrator, you can enable multi-network policy support on your cluster.

Prerequisites

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

- Log in to the cluster with a user with **cluster-admin** privileges.

Procedure

1. Create the **multinetwork-enable-patch.yaml** file with the following YAML:

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  useMultiNetworkPolicy: true
```

2. Configure the cluster to enable multi-network policy:

```
$ oc patch network.operator.openshift.io cluster --type=merge --patch-file=multinetwork-
enable-patch.yaml
```

Example output

```
network.operator.openshift.io/cluster patched
```

25.4.3. Working with multi-network policy

As a cluster administrator, you can create, edit, view, and delete multi-network policies.

25.4.3.1. Prerequisites

- You have enabled multi-network policy support for your cluster.

25.4.3.2. Creating a multi-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 multi-network policy.

Prerequisites

- Your cluster uses a network plugin that supports **NetworkPolicy** objects, such as the OVN-Kubernetes network plugin or the OpenShift SDN network plugin with **mode: NetworkPolicy** set. This mode is the default for OpenShift SDN.

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

- You are logged in to the cluster with a user with **cluster-admin** privileges.

- You are working in the namespace that the multi-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 multi-network policy file name.

b. Define a multi-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.

```
apiVersion: k8s.cni.cncf.io/v1beta1
kind: MultiNetworkPolicy
metadata:
  name: deny-by-default
  annotations:
    k8s.v1.cni.cncf.io/policy-for: <network_name>
spec:
  podSelector:
  ingress: []
```

where:

<network_name>

Specifies the name of a network attachment definition.

Allow ingress from all pods in the same namespace

```
apiVersion: k8s.cni.cncf.io/v1beta1
kind: MultiNetworkPolicy
metadata:
  name: allow-same-namespace
  annotations:
    k8s.v1.cni.cncf.io/policy-for: <network_name>
spec:
  podSelector:
  ingress:
  - from:
    - podSelector: {}
```

where:

<network_name>

Specifies the name of a network attachment definition.

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**.

```
apiVersion: k8s.cni.cncf.io/v1beta1
kind: MultiNetworkPolicy
metadata:
  name: allow-traffic-pod
  annotations:
    k8s.v1.cni.cncf.io/policy-for: <network_name>
spec:
```

```
      podSelector:
       matchLabels:
         pod: pod-a
      policyTypes:
      - Ingress
      ingress:
      - from:
        - namespaceSelector:
            matchLabels:
              kubernetes.io/metadata.name: namespace-y
```

where:

<network_name>

Specifies the name of a network attachment definition.

Restrict traffic to a service

This policy when applied ensures every pod with both labels **app=bookstore** and **role=api** can only be accessed by pods with label **app=bookstore**. In this example the application could be a REST API server, marked with labels **app=bookstore** and **role=api**.

This example addresses the following use cases:

- Restricting the traffic to a service to only the other microservices that need to use it.

- Restricting the connections to a database to only permit the application using it.

```
      apiVersion: k8s.cni.cncf.io/v1beta1
      kind: MultiNetworkPolicy
      metadata:
        name: api-allow
        annotations:
          k8s.v1.cni.cncf.io/policy-for: <network_name>
      spec:
        podSelector:
         matchLabels:
           app: bookstore
           role: api
        ingress:
        - from:
           - podSelector:
              matchLabels:
                app: bookstore
```

where:

<network_name>

Specifies the name of a network attachment definition.

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

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

where:

<policy_name>

Specifies the multi-network policy file name.

<namespace>

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

Example output

> multinetworkpolicy.k8s.cni.cncf.io/deny-by-default created
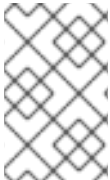
> **NOTE**
>
> If you log in to the web console with **cluster-admin** privileges, you have a choice of creating a network policy in any namespace in the cluster directly in YAML or from a form in the web console.

25.4.3.3. Editing a multi-network policy

You can edit a multi-network policy in a namespace.

Prerequisites

- Your cluster uses a network plugin that supports **NetworkPolicy** objects, such as the OVN-Kubernetes network plugin or the OpenShift SDN network plugin with **mode: NetworkPolicy** set. This mode is the default for OpenShift SDN.

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

- You are logged in to the cluster with a user with **cluster-admin** privileges.

- You are working in the namespace where the multi-network policy exists.

Procedure

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

   > $ oc get multi-networkpolicy

   where:

   <namespace>

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

2. Edit the multi-network policy object.

   - If you saved the multi-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 multi-network policy object directly, enter the following command:

```
$ oc edit multi-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 multi-network policy object is updated.

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

where:

<policy_name>

Specifies the name of the multi-network policy.

<namespace>

Optional: Specifies the namespace if the object is defined in a different namespace than the current namespace.
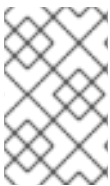
> **NOTE**
>
> If you log in to the web console with **cluster-admin** privileges, you have a choice of editing a network policy in any namespace in the cluster directly in YAML or from the policy in the web console through the **Actions** menu.

25.4.3.4. Viewing multi-network policies using the CLI

You can examine the multi-network policies in a namespace.

Prerequisites

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

- You are logged in to the cluster with a user with **cluster-admin** privileges.

- You are working in the namespace where the multi-network policy exists.

Procedure

- List multi-network policies in a namespace:

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

  ```
  $ oc get multi-networkpolicy
  ```

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

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

  where:

  <policy_name>

  Specifies the name of the multi-network policy to inspect.

  <namespace>

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

> **NOTE**
>
> If you log in to the web console with **cluster-admin** privileges, you have a choice of viewing a network policy in any namespace in the cluster directly in YAML or from a form in the web console.

25.4.3.5. Deleting a multi-network policy using the CLI

You can delete a multi-network policy in a namespace.

Prerequisites

- Your cluster uses a network plugin that supports **NetworkPolicy** objects, such as the OVN-Kubernetes network plugin or the OpenShift SDN network plugin with **mode: NetworkPolicy** set. This mode is the default for OpenShift SDN.

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

- You are logged in to the cluster with a user with **cluster-admin** privileges.

- You are working in the namespace where the multi-network policy exists.

Procedure

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

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

  where:

  <policy_name>

  Specifies the name of the multi-network policy.

  <namespace>

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

Example output

> multinetworkpolicy.k8s.cni.cncf.io/default-deny deleted
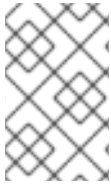
> **NOTE**
>
> If you log in to the web console with **cluster-admin** privileges, you have a choice of deleting a network policy in any namespace in the cluster directly in YAML or from the policy in the web console through the **Actions** menu.

25.4.3.6. Creating a default deny all multi-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.

> **NOTE**
>
> If you log in with a user with the **cluster-admin** role, then you can create a network policy in any namespace in the cluster.

Prerequisites

- Your cluster uses a network plugin that supports **NetworkPolicy** objects, such as the OVN-Kubernetes network plugin or the OpenShift SDN network plugin with **mode: NetworkPolicy** set. This mode is the default for OpenShift SDN.

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

- You are logged in to the cluster with a user with **cluster-admin** privileges.

- You are working in the namespace that the multi-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:

   ```
   apiVersion: k8s.cni.cncf.io/v1beta1
   kind: MultiNetworkPolicy
   metadata:
     name: deny-by-default
     namespace: default 1
     annotations:
       k8s.v1.cni.cncf.io/policy-for: <network_name> 2
   spec:
     podSelector: {} 3
     ingress: [] 4
   ```

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

   2   **network_name**: specifies the name of a network attachment definition.

   3   **podSelector:** is empty, this means it matches all the pods. Therefore, the policy applies to

4 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

```
multinetworkpolicy.k8s.cni.cncf.io/deny-by-default created
```

25.4.3.7. Creating a multi-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**
>
> If you log in with a user with the **cluster-admin** role, then you can create a network policy in any namespace in the cluster.

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

- Your cluster uses a network plugin that supports **NetworkPolicy** objects, such as the OVN-Kubernetes network plugin or the OpenShift SDN network plugin with **mode: NetworkPolicy** set. This mode is the default for OpenShift SDN.

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

- You are logged in to the cluster with a user with **cluster-admin** privileges.

- You are working in the namespace that the multi-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:

```
apiVersion: k8s.cni.cncf.io/v1beta1
kind: MultiNetworkPolicy
metadata:
  name: web-allow-external
  namespace: default
  annotations:
    k8s.v1.cni.cncf.io/policy-for: <network_name>
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

```
multinetworkpolicy.k8s.cni.cncf.io/web-allow-external created
```

This policy allows traffic from all resources, including external traffic as illustrated in the following diagram:

25.4.3.8. Creating a multi-network policy allowing traffic to an application from all namespaces

> **NOTE**
>
> If you log in with a user with the **cluster-admin** role, then you can create a network policy in any namespace in the cluster.

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

Prerequisites

- Your cluster uses a network plugin that supports **NetworkPolicy** objects, such as the OVN-Kubernetes network plugin or the OpenShift SDN network plugin with **mode: NetworkPolicy** set. This mode is the default for OpenShift SDN.

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

- You are logged in to the cluster with a user with **cluster-admin** privileges.

- You are working in the namespace that the multi-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:

```
apiVersion: k8s.cni.cncf.io/v1beta1
kind: MultiNetworkPolicy
metadata:
  name: web-allow-all-namespaces
  namespace: default
  annotations:
    k8s.v1.cni.cncf.io/policy-for: <network_name>
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

```
multinetworkpolicy.k8s.cni.cncf.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>
```

25.4.3.9. Creating a multi-network policy allowing traffic to an application from a namespace

> **NOTE**
>
> If you log in with a user with the **cluster-admin** role, then you can create a network policy in any namespace in the cluster.

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

- Your cluster uses a network plugin that supports **NetworkPolicy** objects, such as the OVN-Kubernetes network plugin or the OpenShift SDN network plugin with **mode: NetworkPolicy** set. This mode is the default for OpenShift SDN.

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

- You are logged in to the cluster with a user with **cluster-admin** privileges.

- You are working in the namespace that the multi-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:

   ```
   apiVersion: k8s.cni.cncf.io/v1beta1
   kind: MultiNetworkPolicy
   metadata:
     name: web-allow-prod
     namespace: default
     annotations:
       k8s.v1.cni.cncf.io/policy-for: <network_name>
   spec:
     podSelector:
       matchLabels:
         app: web ❶
     policyTypes:
     - Ingress
     ingress:
     - from:
       - namespaceSelector:
           matchLabels:
             purpose: production ❷
   ```

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

   ❷   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

   ```
   multinetworkpolicy.k8s.cni.cncf.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>
```

25.4.4. Additional resources

- About network policy

- Understanding multiple networks

- Configuring a macvlan network

- Configuring an SR-IOV network device

25.5. ATTACHING A POD TO AN ADDITIONAL NETWORK

As a cluster user you can attach a pod to an additional network.

25.5.1. Adding a pod to an additional network

You can add a pod to an additional network. The pod continues to send normal cluster-related network traffic over the default network.

When a pod is created additional networks are attached to it. However, if a pod already exists, you cannot attach additional networks to it.

The pod must be in the same namespace as the additional network.

Prerequisites

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

- Log in to the cluster.

Procedure

1. Add an annotation to the **Pod** object. Only one of the following annotation formats can be used:

    a. To attach an additional network without any customization, add an annotation with the following format. Replace **<network>** with the name of the additional network to associate with the pod:

    ```
    metadata:
      annotations:
        k8s.v1.cni.cncf.io/networks: <network>[,<network>,...]
    ```
    ❶

1 To specify more than one additional network, separate each network with a comma. Do not include whitespace between the comma. If you specify the same additional network multiple times, that pod will have multiple network interfaces attached to that network.

b. To attach an additional network with customizations, add an annotation with the following format:

```
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: |-
      [
        {
          "name": "<network>",  1
          "namespace": "<namespace>",  2
          "default-route": ["<default-route>"]  3
        }
      ]
```

1 Specify the name of the additional network defined by a **NetworkAttachmentDefinition** object.

2 Specify the namespace where the **NetworkAttachmentDefinition** object is defined.

3 Optional: Specify an override for the default route, such as **192.168.17.1**.

2. To create the pod, enter the following command. Replace **<name>** with the name of the pod.

```
$ oc create -f <name>.yaml
```

3. Optional: To Confirm that the annotation exists in the **Pod** CR, enter the following command, replacing **<name>** with the name of the pod.

```
$ oc get pod <name> -o yaml
```

In the following example, the **example-pod** pod is attached to the **net1** additional network:

```
$ oc get pod example-pod -o yaml
apiVersion: v1
kind: Pod
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: macvlan-bridge
    k8s.v1.cni.cncf.io/network-status: |-  1
      [{
          "name": "openshift-sdn",
          "interface": "eth0",
          "ips": [
              "10.128.2.14"
          ],
          "default": true,
          "dns": {}
      },{
```

```
        "name": "macvlan-bridge",
        "interface": "net1",
        "ips": [
            "20.2.2.100"
        ],
        "mac": "22:2f:60:a5:f8:00",
        "dns": {}
    }]
  name: example-pod
  namespace: default
spec:
  ...
status:
  ...
```

1   The **k8s.v1.cni.cncf.io/network-status** parameter is a JSON array of objects. Each object describes the status of an additional network attached to the pod. The annotation value is stored as a plain text value.

25.5.1.1. Specifying pod-specific addressing and routing options

When attaching a pod to an additional network, you may want to specify further properties about that network in a particular pod. This allows you to change some aspects of routing, as well as specify static IP addresses and MAC addresses. To accomplish this, you can use the JSON formatted annotations.

Prerequisites

- The pod must be in the same namespace as the additional network.

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

- You must log in to the cluster.

Procedure

To add a pod to an additional network while specifying addressing and/or routing options, complete the following steps:

1. Edit the **Pod** resource definition. If you are editing an existing **Pod** resource, run the following command to edit its definition in the default editor. Replace **<name>** with the name of the **Pod** resource to edit.

   ```
   $ oc edit pod <name>
   ```

2. In the **Pod** resource definition, add the **k8s.v1.cni.cncf.io/networks** parameter to the pod **metadata** mapping. The **k8s.v1.cni.cncf.io/networks** accepts a JSON string of a list of objects that reference the name of **NetworkAttachmentDefinition** custom resource (CR) names in addition to specifying additional properties.

   ```
   metadata:
     annotations:
       k8s.v1.cni.cncf.io/networks: '[<network>[,<network>,...]]' 1
   ```

[1] Replace **<network>** with a JSON object as shown in the following examples. The single quotes are required.

3. In the following example the annotation specifies which network attachment will have the default route, using the **default-route** parameter.

```
apiVersion: v1
kind: Pod
metadata:
  name: example-pod
  annotations:
    k8s.v1.cni.cncf.io/networks: '[
    {
      "name": "net1"
    },
    {
      "name": "net2", 1
      "default-route": ["192.0.2.1"] 2
    }]'
spec:
  containers:
  - name: example-pod
    command: ["/bin/bash", "-c", "sleep 2000000000000"]
    image: centos/tools
```

[1] The **name** key is the name of the additional network to associate with the pod.

[2] The **default-route** key specifies a value of a gateway for traffic to be routed over if no other routing entry is present in the routing table. If more than one **default-route** key is specified, this will cause the pod to fail to become active.

The default route will cause any traffic that is not specified in other routes to be routed to the gateway.

> **IMPORTANT**
>
> Setting the default route to an interface other than the default network interface for OpenShift Container Platform may cause traffic that is anticipated for pod-to-pod traffic to be routed over another interface.

To verify the routing properties of a pod, the **oc** command may be used to execute the **ip** command within a pod.

```
$ oc exec -it <pod_name> -- ip route
```

> **NOTE**
>
> You may also reference the pod's **k8s.v1.cni.cncf.io/network-status** to see which additional network has been assigned the default route, by the presence of the **default-route** key in the JSON-formatted list of objects.

To set a static IP address or MAC address for a pod you can use the JSON formatted annotations. This requires you create networks that specifically allow for this functionality. This can be specified in a rawCNIConfig for the CNO.

1. Edit the CNO CR by running the following command:

```
$ oc edit networks.operator.openshift.io cluster
```

The following YAML describes the configuration parameters for the CNO:

Cluster Network Operator YAML configuration

```
name: <name> 1
namespace: <namespace> 2
rawCNIConfig: '{ 3
  ...
}'
type: Raw
```

1 Specify a name for the additional network attachment that you are creating. The name must be unique within the specified **namespace**.

2 Specify the namespace to create the network attachment in. If you do not specify a value, then the **default** namespace is used.

3 Specify the CNI plugin configuration in JSON format, which is based on the following template.

The following object describes the configuration parameters for utilizing static MAC address and IP address using the macvlan CNI plugin:

macvlan CNI plugin JSON configuration object using static IP and MAC address

```
{
  "cniVersion": "0.3.1",
  "name": "<name>", 1
  "plugins": [{ 2
    "type": "macvlan",
    "capabilities": { "ips": true }, 3
    "master": "eth0", 4
    "mode": "bridge",
    "ipam": {
      "type": "static"
    }
  }, {
    "capabilities": { "mac": true }, 5
    "type": "tuning"
  }]
}
```

1 Specifies the name for the additional network attachment to create. The name must be unique within the specified **namespace**.

2

Specifies an array of CNI plugin configurations. The first object specifies a macvlan plugin configuration and the second object specifies a tuning plugin configuration.

3 Specifies that a request is made to enable the static IP address functionality of the CNI plugin runtime configuration capabilities.

4 Specifies the interface that the macvlan plugin uses.

5 Specifies that a request is made to enable the static MAC address functionality of a CNI plugin.

The above network attachment can be referenced in a JSON formatted annotation, along with keys to specify which static IP and MAC address will be assigned to a given pod.

Edit the pod with:

```
$ oc edit pod <name>
```

macvlan CNI plugin JSON configuration object using static IP and MAC address

```
apiVersion: v1
kind: Pod
metadata:
  name: example-pod
  annotations:
    k8s.v1.cni.cncf.io/networks: '[
      {
        "name": "<name>", 1
        "ips": [ "192.0.2.205/24" ], 2
        "mac": "CA:FE:C0:FF:EE:00" 3
      }
    ]'
```

1 Use the **<name>** as provided when creating the **rawCNIConfig** above.

2 Provide an IP address including the subnet mask.

3 Provide the MAC address.

> **NOTE**
>
> Static IP addresses and MAC addresses do not have to be used at the same time, you may use them individually, or together.

To verify the IP address and MAC properties of a pod with additional networks, use the **oc** command to execute the ip command within a pod.

```
$ oc exec -it <pod_name> -- ip a
```

25.6. REMOVING A POD FROM AN ADDITIONAL NETWORK

As a cluster user you can remove a pod from an additional network.

25.6.1. Removing a pod from an additional network

You can remove a pod from an additional network only by deleting the pod.

Prerequisites

- An additional network is attached to the pod.

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

- Log in to the cluster.

Procedure

- To delete the pod, enter the following command:

```
$ oc delete pod <name> -n <namespace>
```

  ○ **<name>** is the name of the pod.

  ○ **<namespace>** is the namespace that contains the pod.

25.7. EDITING AN ADDITIONAL NETWORK

As a cluster administrator you can modify the configuration for an existing additional network.

25.7.1. Modifying an additional network attachment definition

As a cluster administrator, you can make changes to an existing additional network. Any existing pods attached to the additional network will not be updated.

Prerequisites

- You have configured an additional network for your cluster.

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

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

Procedure

To edit an additional network for your cluster, complete the following steps:

1. Run the following command to edit the Cluster Network Operator (CNO) CR in your default text editor:

```
$ oc edit networks.operator.openshift.io cluster
```

2. In the **additionalNetworks** collection, update the additional network with your changes.

3. Save your changes and quit the text editor to commit your changes.

4. Optional: Confirm that the CNO updated the **NetworkAttachmentDefinition** object by running the following command. Replace **<network-name>** with the name of the additional network to display. There might be a delay before the CNO updates the **NetworkAttachmentDefinition**

object to reflect your changes.

```
$ oc get network-attachment-definitions <network-name> -o yaml
```

For example, the following console output displays a **NetworkAttachmentDefinition** object that is named **net1**:

```
$ oc get network-attachment-definitions net1 -o go-template='{{printf "%s\n" .spec.config}}'
{ "cniVersion": "0.3.1", "type": "macvlan",
"master": "ens5",
"mode": "bridge",
"ipam":      {"type":"static","routes":[{"dst":"0.0.0.0/0","gw":"10.128.2.1"}],"addresses":
[{"address":"10.128.2.100/23","gateway":"10.128.2.1"}],"dns":{"nameservers":
["172.30.0.10"],"domain":"us-west-2.compute.internal","search":["us-west-
2.compute.internal"]}} }
```

25.8. REMOVING AN ADDITIONAL NETWORK

As a cluster administrator you can remove an additional network attachment.

25.8.1. Removing an additional network attachment definition

As a cluster administrator, you can remove an additional network from your OpenShift Container Platform cluster. The additional network is not removed from any pods it is attached to.

Prerequisites

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

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

Procedure

To remove an additional network from your cluster, complete the following steps:

1. Edit the Cluster Network Operator (CNO) in your default text editor by running the following command:

   ```
   $ oc edit networks.operator.openshift.io cluster
   ```

2. Modify the CR by removing the configuration from the **additionalNetworks** collection for the network attachment definition you are removing.

   ```
   apiVersion: operator.openshift.io/v1
   kind: Network
   metadata:
     name: cluster
   spec:
     additionalNetworks: []  1
   ```

   1   If you are removing the configuration mapping for the only additional network attachment definition in the **additionalNetworks** collection, you must specify an empty collection.

3. Save your changes and quit the text editor to commit your changes.

4. Optional: Confirm that the additional network CR was deleted by running the following command:

```
$ oc get network-attachment-definition --all-namespaces
```

25.9. ASSIGNING A SECONDARY NETWORK TO A VRF

As a cluster administrator, you can configure an additional network for a virtual routing and forwarding (VRF) domain by using the CNI VRF plugin. The virtual network that this plugin creates is associated with the physical interface that you specify.

Using a secondary network with a VRF instance has the following advantages:

Workload isolation

Isolate workload traffic by configuring a VRF instance for the additional network.

Improved security

Enable improved security through isolated network paths in the VRF domain.

Multi-tenancy support

Support multi-tenancy through network segmentation with a unique routing table in the VRF domain for each tenant.

> **NOTE**
>
> Applications that use VRFs must bind to a specific device. The common usage is to use the **SO_BINDTODEVICE** option for a socket. The **SO_BINDTODEVICE** option binds the socket to the device that is specified in the passed interface name, for example, **eth1**. To use the **SO_BINDTODEVICE** option, the application must have **CAP_NET_RAW** capabilities.
>
> Using a VRF through the **ip vrf exec** command is not supported in OpenShift Container Platform pods. To use VRF, bind applications directly to the VRF interface.

Additional resources

- [About virtual routing and forwarding](#)

25.9.1. Creating an additional network attachment with the CNI VRF plugin

The Cluster Network Operator (CNO) manages additional network definitions. When you specify an additional network to create, the CNO creates the **NetworkAttachmentDefinition** custom resource (CR) automatically.

> **NOTE**
>
> Do not edit the **NetworkAttachmentDefinition** CRs that the Cluster Network Operator manages. Doing so might disrupt network traffic on your additional network.

To create an additional network attachment with the CNI VRF plugin, perform the following procedure.

Prerequisites

- Install the OpenShift Container Platform CLI (oc).

- Log in to the OpenShift cluster as a user with cluster-admin privileges.

Procedure

1. Create the **Network** custom resource (CR) for the additional network attachment and insert the **rawCNIConfig** configuration for the additional network, as in the following example CR. Save the YAML as the file **additional-network-attachment.yaml**.

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  additionalNetworks:
   - name: test-network-1
     namespace: additional-network-1
     type: Raw
     rawCNIConfig: '{
      "cniVersion": "0.3.1",
      "name": "macvlan-vrf",
      "plugins": [        1
      {
        "type": "macvlan",
        "master": "eth1",
        "ipam": {
           "type": "static",
           "addresses": [
           {
               "address": "191.168.1.23/24"
           }
           ]
        }
      },
      {
        "type": "vrf",        2
        "vrfname": "vrf-1",        3
        "table": 1001        4
      }]
     }'
```

1   **plugins** must be a list. The first item in the list must be the secondary network underpinning the VRF network. The second item in the list is the VRF plugin configuration.

2   **type** must be set to **vrf**.

3   **vrfname** is the name of the VRF that the interface is assigned to. If it does not exist in the pod, it is created.

4   Optional. **table** is the routing table ID. By default, the **tableid** parameter is used. If it is not specified, the CNI assigns a free routing table ID to the VRF.
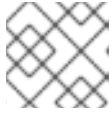
> **NOTE**
>
> VRF functions correctly only when the resource is of type **netdevice**.

2. Create the **Network** resource:

   ```
   $ oc create -f additional-network-attachment.yaml
   ```

3. Confirm that the CNO created the **NetworkAttachmentDefinition** CR by running the following command. Replace **<namespace>** with the namespace that you specified when configuring the network attachment, for example, **additional-network-1**.

   ```
   $ oc get network-attachment-definitions -n <namespace>
   ```

   Example output

   ```
   NAME                  AGE
   additional-network-1     14m
   ```
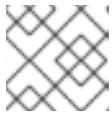
> **NOTE**
>
> There might be a delay before the CNO creates the CR.

Verification

1. Create a pod and assign it to the additional network with the VRF instance:

   a. Create a YAML file that defines the **Pod** resource:

   Example pod-additional-net.yaml file

   ```
   apiVersion: v1
   kind: Pod
   metadata:
    name: pod-additional-net
    annotations:
      k8s.v1.cni.cncf.io/networks: '[
        {
            "name": "test-network-1" ❶
        }
    ]'
   spec:
    containers:
    - name: example-pod-1
      command: ["/bin/bash", "-c", "sleep 9000000"]
      image: centos:8
   ```

   ❶ Specify the name of the additional network with the VRF instance.

   b. Create the **Pod** resource by running the following command:

   ```
   $ oc create -f pod-additional-net.yaml
   ```

Example output

```
pod/test-pod created
```

2. Verify that the pod network attachment is connected to the VRF additional network. Start a remote session with the pod and run the following command:

```
$ ip vrf show
```

Example output

```
Name              Table
-----------------------
vrf-1             1001
```

3. Confirm that the VRF interface is the controller for the additional interface:

```
$ ip link
```

Example output

```
5: net1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master red
state UP mode
```

CHAPTER 26. HARDWARE NETWORKS

26.1. ABOUT SINGLE ROOT I/O VIRTUALIZATION (SR-IOV) HARDWARE NETWORKS

The Single Root I/O Virtualization (SR-IOV) specification is a standard for a type of PCI device assignment that can share a single device with multiple pods.

SR-IOV can segment a compliant network device, recognized on the host node as a physical function (PF), into multiple virtual functions (VFs). The VF is used like any other network device. The SR-IOV network device driver for the device determines how the VF is exposed in the container:

- **netdevice** driver: A regular kernel network device in the **netns** of the container

- **vfio-pci** driver: A character device mounted in the container

You can use SR-IOV network devices with additional networks on your OpenShift Container Platform cluster installed on bare metal or Red Hat OpenStack Platform (RHOSP) infrastructure for applications that require high bandwidth or low latency.

You can configure multi-network policies for SR-IOV networks. The support for this is technology preview and SR-IOV additional networks are only supported with kernel NICs. They are not supported for Data Plane Development Kit (DPDK) applications.

> **NOTE**
>
> Creating multi-network policies on SR-IOV networks might not deliver the same performance to applications compared to SR-IOV networks without a multi-network policy configured.
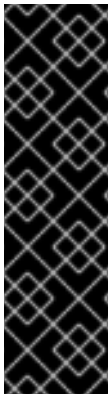
> **IMPORTANT**
>
> Multi-network policies for SR-IOV network is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.
>
> For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

You can enable SR-IOV on a node by using the following command:

```
$ oc label node <node_name> feature.node.kubernetes.io/network-sriov.capable="true"
```

26.1.1. Components that manage SR-IOV network devices

The SR-IOV Network Operator creates and manages the components of the SR-IOV stack. It performs the following functions:

- Orchestrates discovery and management of SR-IOV network devices

- Generates **NetworkAttachmentDefinition** custom resources for the SR-IOV Container Network Interface (CNI)

- Creates and updates the configuration of the SR-IOV network device plugin

- Creates node specific **SriovNetworkNodeState** custom resources

- Updates the **spec.interfaces** field in each **SriovNetworkNodeState** custom resource

The Operator provisions the following components:

SR-IOV network configuration daemon

A daemon set that is deployed on worker nodes when the SR-IOV Network Operator starts. The daemon is responsible for discovering and initializing SR-IOV network devices in the cluster.

SR-IOV Network Operator webhook

A dynamic admission controller webhook that validates the Operator custom resource and sets appropriate default values for unset fields.

SR-IOV Network resources injector

A dynamic admission controller webhook that provides functionality for patching Kubernetes pod specifications with requests and limits for custom network resources such as SR-IOV VFs. The SR-IOV network resources injector adds the **resource** field to only the first container in a pod automatically.

SR-IOV network device plugin

A device plugin that discovers, advertises, and allocates SR-IOV network virtual function (VF) resources. Device plugins are used in Kubernetes to enable the use of limited resources, typically in physical devices. Device plugins give the Kubernetes scheduler awareness of resource availability, so that the scheduler can schedule pods on nodes with sufficient resources.

SR-IOV CNI plugin

A CNI plugin that attaches VF interfaces allocated from the SR-IOV network device plugin directly into a pod.

SR-IOV InfiniBand CNI plugin

A CNI plugin that attaches InfiniBand (IB) VF interfaces allocated from the SR-IOV network device plugin directly into a pod.

> **NOTE**
>
> The SR-IOV Network resources injector and SR-IOV Network Operator webhook are enabled by default and can be disabled by editing the **default SriovOperatorConfig** CR. Use caution when disabling the SR-IOV Network Operator Admission Controller webhook. You can disable the webhook under specific circumstances, such as troubleshooting, or if you want to use unsupported devices.

26.1.1.1. Supported platforms

The SR-IOV Network Operator is supported on the following platforms:

- Bare metal

- Red Hat OpenStack Platform (RHOSP)

26.1.1.2. Supported devices

OpenShift Container Platform supports the following network interface controllers:

Table 26.1. Supported network interface controllers

Manufacturer	Model	Vendor ID	Device ID
Broadcom	BCM57414	14e4	16d7
Broadcom	BCM57508	14e4	1750
Broadcom	BCM57504	14e4	1751
Intel	X710	8086	1572
Intel	XL710	8086	1583
Intel	X710 Base T	8086	15ff
Intel	XXV710	8086	158b
Intel	E810-CQDA2	8086	1592
Intel	E810-2CQDA2	8086	1592
Intel	E810-XXVDA2	8086	159b
Intel	E810-XXVDA4	8086	1593
Intel	E810-XXVDA4T	8086	1593
Mellanox	MT27700 Family [ConnectX-4]	15b3	1013
Mellanox	MT27710 Family [ConnectX-4 Lx]	15b3	1015
Mellanox	MT27800 Family [ConnectX-5]	15b3	1017
Mellanox	MT28880 Family [ConnectX-5 Ex]	15b3	1019
Mellanox	MT28908 Family [ConnectX-6]	15b3	101b
Mellanox	MT2892 Family [ConnectX-6 Dx]	15b3	101d
Mellanox	MT2894 Family [ConnectX-6 Lx]	15b3	101f
Mellanox	MT42822 BlueField-2 in ConnectX-6 NIC mode	15b3	a2d6

Manufacturer	Model	Vendor ID	Device ID
Pensando [1]	DSC-25 dual-port 25G distributed services card for ionic driver	0x1dd8	0x1002
Pensando [1]	DSC-100 dual-port 100G distributed services card for ionic driver	0x1dd8	0x1003
Silicom	STS Family	8086	1591

1. OpenShift SR-IOV is supported, but you must set a static, Virtual Function (VF) media access control (MAC) address using the SR-IOV CNI config file when using SR-IOV.
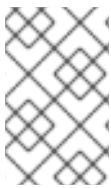
> **NOTE**
>
> For the most up-to-date list of supported cards and compatible OpenShift Container Platform versions available, see Openshift Single Root I/O Virtualization (SR-IOV) and PTP hardware networks Support Matrix.

26.1.1.3. Automated discovery of SR-IOV network devices

The SR-IOV Network Operator searches your cluster for SR-IOV capable network devices on worker nodes. The Operator creates and updates a SriovNetworkNodeState custom resource (CR) for each worker node that provides a compatible SR-IOV network device.

The CR is assigned the same name as the worker node. The **status.interfaces** list provides information about the network devices on a node.
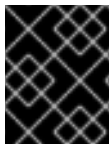
> **IMPORTANT**
>
> Do not modify a **SriovNetworkNodeState** object. The Operator creates and manages these resources automatically.

26.1.1.3.1. Example SriovNetworkNodeState object

The following YAML is an example of a **SriovNetworkNodeState** object created by the SR-IOV Network Operator:

An SriovNetworkNodeState object

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodeState
metadata:
  name: node-25 1
  namespace: openshift-sriov-network-operator
  ownerReferences:
  - apiVersion: sriovnetwork.openshift.io/v1
    blockOwnerDeletion: true
    controller: true
    kind: SriovNetworkNodePolicy
    name: default
spec:
```

```
    dpConfigVersion: "39824"
  status:
   interfaces: 2
   - deviceID: "1017"
     driver: mlx5_core
     mtu: 1500
     name: ens785f0
     pciAddress: "0000:18:00.0"
     totalvfs: 8
     vendor: 15b3
   - deviceID: "1017"
     driver: mlx5_core
     mtu: 1500
     name: ens785f1
     pciAddress: "0000:18:00.1"
     totalvfs: 8
     vendor: 15b3
   - deviceID: 158b
     driver: i40e
     mtu: 1500
     name: ens817f0
     pciAddress: 0000:81:00.0
     totalvfs: 64
     vendor: "8086"
   - deviceID: 158b
     driver: i40e
     mtu: 1500
     name: ens817f1
     pciAddress: 0000:81:00.1
     totalvfs: 64
     vendor: "8086"
   - deviceID: 158b
     driver: i40e
     mtu: 1500
     name: ens803f0
     pciAddress: 0000:86:00.0
     totalvfs: 64
     vendor: "8086"
  syncStatus: Succeeded
```

1. The value of the **name** field is the same as the name of the worker node.

2. The **interfaces** stanza includes a list of all of the SR-IOV devices discovered by the Operator on the worker node.

26.1.1.4. Example use of a virtual function in a pod

You can run a remote direct memory access (RDMA) or a Data Plane Development Kit (DPDK) application in a pod with SR-IOV VF attached.

This example shows a pod using a virtual function (VF) in RDMA mode:

Pod spec that uses RDMA mode

```
apiVersion: v1
kind: Pod
metadata:
  name: rdma-app
  annotations:
    k8s.v1.cni.cncf.io/networks: sriov-rdma-mlnx
spec:
  containers:
  - name: testpmd
    image: <RDMA_image>
    imagePullPolicy: IfNotPresent
    securityContext:
      runAsUser: 0
      capabilities:
        add: ["IPC_LOCK","SYS_RESOURCE","NET_RAW"]
    command: ["sleep", "infinity"]
```

The following example shows a pod with a VF in DPDK mode:

Pod spec that uses DPDK mode

```
apiVersion: v1
kind: Pod
metadata:
  name: dpdk-app
  annotations:
    k8s.v1.cni.cncf.io/networks: sriov-dpdk-net
spec:
  containers:
  - name: testpmd
    image: <DPDK_image>
    securityContext:
      runAsUser: 0
      capabilities:
        add: ["IPC_LOCK","SYS_RESOURCE","NET_RAW"]
    volumeMounts:
    - mountPath: /dev/hugepages
      name: hugepage
    resources:
      limits:
        memory: "1Gi"
        cpu: "2"
        hugepages-1Gi: "4Gi"
      requests:
        memory: "1Gi"
        cpu: "2"
        hugepages-1Gi: "4Gi"
    command: ["sleep", "infinity"]
  volumes:
  - name: hugepage
    emptyDir:
      medium: HugePages
```

26.1.1.5. DPDK library for use with container applications

An optional library, **app-netutil**, provides several API methods for gathering network information about a pod from within a container running within that pod.

This library can assist with integrating SR-IOV virtual functions (VFs) in Data Plane Development Kit (DPDK) mode into the container. The library provides both a Golang API and a C API.

Currently there are three API methods implemented:

GetCPUInfo()

> This function determines which CPUs are available to the container and returns the list.

GetHugepages()

> This function determines the amount of huge page memory requested in the **Pod** spec for each container and returns the values.

GetInterfaces()

> This function determines the set of interfaces in the container and returns the list. The return value includes the interface type and type-specific data for each interface.

The repository for the library includes a sample Dockerfile to build a container image, **dpdk-app-centos**. The container image can run one of the following DPDK sample applications, depending on an environment variable in the pod specification: **l2fwd**, **l3wd** or **testpmd**. The container image provides an example of integrating the **app-netutil** library into the container image itself. The library can also integrate into an init container. The init container can collect the required data and pass the data to an existing DPDK workload.

26.1.1.6. Huge pages resource injection for Downward API

When a pod specification includes a resource request or limit for huge pages, the Network Resources Injector automatically adds Downward API fields to the pod specification to provide the huge pages information to the container.

The Network Resources Injector adds a volume that is named **podnetinfo** and is mounted at /**etc/podnetinfo** for each container in the pod. The volume uses the Downward API and includes a file for huge pages requests and limits. The file naming convention is as follows:

- /**etc/podnetinfo/hugepages_1G_request_<container-name>**

- /**etc/podnetinfo/hugepages_1G_limit_<container-name>**

- /**etc/podnetinfo/hugepages_2M_request_<container-name>**

- /**etc/podnetinfo/hugepages_2M_limit_<container-name>**

The paths specified in the previous list are compatible with the **app-netutil** library. By default, the library is configured to search for resource information in the /**etc/podnetinfo** directory. If you choose to specify the Downward API path items yourself manually, the **app-netutil** library searches for the following paths in addition to the paths in the previous list.

- /**etc/podnetinfo/hugepages_request**

- /**etc/podnetinfo/hugepages_limit**

- /**etc/podnetinfo/hugepages_1G_request**

- /**etc/podnetinfo/hugepages_1G_limit**

- **/etc/podnetinfo/hugepages_2M_request**

- **/etc/podnetinfo/hugepages_2M_limit**

As with the paths that the Network Resources Injector can create, the paths in the preceding list can optionally end with a **_<container-name>** suffix.

26.1.2. Additional resources

- Configuring multi-network policy

26.1.3. Next steps

- Installing the SR-IOV Network Operator

- Optional: Configuring the SR-IOV Network Operator

- Configuring an SR-IOV network device

- If you use OpenShift Virtualization: Connecting a virtual machine to an SR-IOV network

- Configuring an SR-IOV network attachment

- Adding a pod to an SR-IOV additional network

26.2. INSTALLING THE SR-IOV NETWORK OPERATOR

You can install the Single Root I/O Virtualization (SR-IOV) Network Operator on your cluster to manage SR-IOV network devices and network attachments.

26.2.1. Installing SR-IOV Network Operator

As a cluster administrator, you can install the SR-IOV Network Operator by using the OpenShift Container Platform CLI or the web console.

26.2.1.1. CLI: Installing the SR-IOV Network Operator

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

Prerequisites

- A cluster installed on bare-metal hardware with nodes that have hardware that supports SR-IOV.

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

- An account with **cluster-admin** privileges.

Procedure

1. To create the **openshift-sriov-network-operator** namespace, enter the following command:

   ```
   $ cat << EOF| oc create -f -
   apiVersion: v1
   ```

```
kind: Namespace
metadata:
  name: openshift-sriov-network-operator
  annotations:
    workload.openshift.io/allowed: management
EOF
```

2. To create an OperatorGroup CR, enter the following command:

```
$ cat << EOF| oc create -f -
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: sriov-network-operators
  namespace: openshift-sriov-network-operator
spec:
  targetNamespaces:
  - openshift-sriov-network-operator
EOF
```

3. Subscribe to the SR-IOV Network Operator.

   a. Run the following command to get the OpenShift Container Platform major and minor version. It is required for the **channel** value in the next step.

   ```
   $ OC_VERSION=$(oc version -o yaml | grep openshiftVersion | \
       grep -o '[0-9]*[.][0-9]*' | head -1)
   ```

   b. To create a Subscription CR for the SR-IOV Network Operator, enter the following command:

   ```
   $ cat << EOF| oc create -f -
   apiVersion: operators.coreos.com/v1alpha1
   kind: Subscription
   metadata:
     name: sriov-network-operator-subscription
     namespace: openshift-sriov-network-operator
   spec:
     channel: "${OC_VERSION}"
     name: sriov-network-operator
     source: redhat-operators
     sourceNamespace: openshift-marketplace
   EOF
   ```

4. To verify that the Operator is installed, enter the following command:

   ```
   $ oc get csv -n openshift-sriov-network-operator \
     -o custom-columns=Name:.metadata.name,Phase:.status.phase
   ```

   Example output

   ```
   Name                                 Phase
   sriov-network-operator.4.13.0-202310121402   Succeeded
   ```

26.2.1.2. Web console: Installing the SR-IOV Network Operator

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

Prerequisites

- A cluster installed on bare-metal hardware with nodes that have hardware that supports SR-IOV.

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

- An account with **cluster-admin** privileges.

Procedure

1. Install the SR-IOV Network Operator:

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

   b. Select **SR-IOV Network Operator** from the list of available Operators, and then click **Install**.

   c. On the **Install Operator** page, under **Installed Namespace**, select **Operator recommended Namespace**.

   d. Click **Install**.

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

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

   b. Ensure that **SR-IOV Network Operator** is listed in the **openshift-sriov-network-operator** project with a **Status** of **InstallSucceeded**.

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

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

   - Inspect the **Operator Subscriptions** and **Install Plans** tabs for any failure or errors under **Status**.

   - Navigate to the **Workloads → Pods** page and check the logs for pods in the **openshift-sriov-network-operator** project.

   - Check the namespace of the YAML file. If the annotation is missing, you can add the annotation **workload.openshift.io/allowed=management** to the Operator namespace with the following command:

     ```
     $ oc annotate ns/openshift-sriov-network-operator
     workload.openshift.io/allowed=management
     ```

NOTE

For single-node OpenShift clusters, the annotation **workload.openshift.io/allowed=management** is required for the namespace.

26.2.2. Next steps

- Optional: Configuring the SR-IOV Network Operator

26.3. CONFIGURING THE SR-IOV NETWORK OPERATOR

The Single Root I/O Virtualization (SR-IOV) Network Operator manages the SR-IOV network devices and network attachments in your cluster.

26.3.1. Configuring the SR-IOV Network Operator
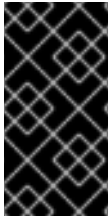
IMPORTANT

Modifying the SR-IOV Network Operator configuration is not normally necessary. The default configuration is recommended for most use cases. Complete the steps to modify the relevant configuration only if the default behavior of the Operator is not compatible with your use case.

The SR-IOV Network Operator adds the **SriovOperatorConfig.sriovnetwork.openshift.io** CustomResourceDefinition resource. The Operator automatically creates a SriovOperatorConfig custom resource (CR) named **default** in the **openshift-sriov-network-operator** namespace.

NOTE

The **default** CR contains the SR-IOV Network Operator configuration for your cluster. To change the Operator configuration, you must modify this CR.

26.3.1.1. SR-IOV Network Operator config custom resource

The fields for the **sriovoperatorconfig** custom resource are described in the following table:

Table 26.2. SR-IOV Network Operator config custom resource

Field	Type	Description
metadata.name	**string**	Specifies the name of the SR-IOV Network Operator instance. The default value is **default**. Do not set a different value.
metadata.name space	**string**	Specifies the namespace of the SR-IOV Network Operator instance. The default value is **openshift-sriov-network-operator**. Do not set a different value.
spec.configDae monNodeSelect or	**string**	Specifies the node selection to control scheduling the SR-IOV Network Config Daemon on selected nodes. By default, this field is not set and the Operator deploys the SR-IOV Network Config daemon set on worker nodes.

Field	Type	Description
spec.disableDrain	**boolean**	Specifies whether to disable the node draining process or enable the node draining process when you apply a new policy to configure the NIC on a node. Setting this field to **true** facilitates software development and installing OpenShift Container Platform on a single node. By default, this field is not set. For single-node clusters, set this field to **true** after installing the Operator. This field must remain set to **true**.
spec.enableInjector	**boolean**	Specifies whether to enable or disable the Network Resources Injector daemon set. By default, this field is set to **true**.
spec.enableOperatorWebhook	**boolean**	Specifies whether to enable or disable the Operator Admission Controller webhook daemon set. By default, this field is set to **true**.
spec.logLevel	**integer**	Specifies the log verbosity level of the Operator. Set to **0** to show only the basic logs. Set to **2** to show all the available logs. By default, this field is set to **2**.

26.3.1.2. About the Network Resources Injector

The Network Resources Injector is a Kubernetes Dynamic Admission Controller application. It provides the following capabilities:

- Mutation of resource requests and limits in a pod specification to add an SR-IOV resource name according to an SR-IOV network attachment definition annotation.

- Mutation of a pod specification with a Downward API volume to expose pod annotations, labels, and huge pages requests and limits. Containers that run in the pod can access the exposed information as files under the **/etc/podnetinfo** path.

By default, the Network Resources Injector is enabled by the SR-IOV Network Operator and runs as a daemon set on all control plane nodes. The following is an example of Network Resources Injector pods running in a cluster with three control plane nodes:

```
$ oc get pods -n openshift-sriov-network-operator
```

Example output

```
NAME                             READY  STATUS   RESTARTS  AGE
network-resources-injector-5cz5p        1/1    Running  0        10m
network-resources-injector-dwqpx        1/1    Running  0        10m
network-resources-injector-lktz5        1/1    Running  0        10m
```

26.3.1.3. About the SR-IOV Network Operator admission controller webhook

The SR-IOV Network Operator Admission Controller webhook is a Kubernetes Dynamic Admission Controller application. It provides the following capabilities:

- Validation of the **SriovNetworkNodePolicy** CR when it is created or updated.

- Mutation of the **SriovNetworkNodePolicy** CR by setting the default value for the **priority** and **deviceType** fields when the CR is created or updated.

By default the SR-IOV Network Operator Admission Controller webhook is enabled by the Operator and runs as a daemon set on all control plane nodes.

> **NOTE**
>
> Use caution when disabling the SR-IOV Network Operator Admission Controller webhook. You can disable the webhook under specific circumstances, such as troubleshooting, or if you want to use unsupported devices. For information about configuring unsupported devices, see Configuring the SR-IOV Network Operator to use an unsupported NIC.

The following is an example of the Operator Admission Controller webhook pods running in a cluster with three control plane nodes:

```
$ oc get pods -n openshift-sriov-network-operator
```

Example output

```
NAME                        READY  STATUS   RESTARTS  AGE
operator-webhook-9jkw6        1/1    Running  0         16m
operator-webhook-kbr5p        1/1    Running  0         16m
operator-webhook-rpfrl        1/1    Running  0         16m
```

26.3.1.4. About custom node selectors

The SR-IOV Network Config daemon discovers and configures the SR-IOV network devices on cluster nodes. By default, it is deployed to all the **worker** nodes in the cluster. You can use node labels to specify on which nodes the SR-IOV Network Config daemon runs.

26.3.1.5. Disabling or enabling the Network Resources Injector

To disable or enable the Network Resources Injector, which is enabled by default, complete the following procedure.

Prerequisites

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

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

- You must have installed the SR-IOV Network Operator.

Procedure

- Set the **enableInjector** field. Replace **<value>** with **false** to disable the feature or **true** to enable the feature.

```
$ oc patch sriovoperatorconfig default \
  --type=merge -n openshift-sriov-network-operator \
  --patch '{ "spec": { "enableInjector": <value> } }'
```

TIP

You can alternatively apply the following YAML to update the Operator:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovOperatorConfig
metadata:
  name: default
  namespace: openshift-sriov-network-operator
spec:
  enableInjector: <value>
```

26.3.1.6. Disabling or enabling the SR-IOV Network Operator admission controller webhook

To disable or enable the admission controller webhook, which is enabled by default, complete the following procedure.

Prerequisites

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

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

- You must have installed the SR-IOV Network Operator.

Procedure

- Set the **enableOperatorWebhook** field. Replace **<value>** with **false** to disable the feature or **true** to enable it:

```
$ oc patch sriovoperatorconfig default --type=merge \
  -n openshift-sriov-network-operator \
  --patch '{ "spec": { "enableOperatorWebhook": <value> } }'
```

TIP

You can alternatively apply the following YAML to update the Operator:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovOperatorConfig
metadata:
  name: default
  namespace: openshift-sriov-network-operator
spec:
  enableOperatorWebhook: <value>
```

26.3.1.7. Configuring a custom NodeSelector for the SR-IOV Network Config daemon

The SR-IOV Network Config daemon discovers and configures the SR-IOV network devices on cluster nodes. By default, it is deployed to all the **worker** nodes in the cluster. You can use node labels to specify on which nodes the SR-IOV Network Config daemon runs.

To specify the nodes where the SR-IOV Network Config daemon is deployed, complete the following procedure.

> **IMPORTANT**
>
> When you update the **configDaemonNodeSelector** field, the SR-IOV Network Config daemon is recreated on each selected node. While the daemon is recreated, cluster users are unable to apply any new SR-IOV Network node policy or create new SR-IOV pods.

Procedure

- To update the node selector for the operator, enter the following command:

```
$ oc patch sriovoperatorconfig default --type=json \
  -n openshift-sriov-network-operator \
  --patch '[{
    "op": "replace",
    "path": "/spec/configDaemonNodeSelector",
    "value": {<node_label>}
  }]'
```

  Replace **<node_label>** with a label to apply as in the following example: **"node-role.kubernetes.io/worker": ""**.

  TIP

  You can alternatively apply the following YAML to update the Operator:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovOperatorConfig
metadata:
  name: default
  namespace: openshift-sriov-network-operator
spec:
  configDaemonNodeSelector:
    <node_label>
```

26.3.1.8. Configuring the SR-IOV Network Operator for single node installations

By default, the SR-IOV Network Operator drains workloads from a node before every policy change. The Operator performs this action to ensure that there no workloads using the virtual functions before the reconfiguration.

For installations on a single node, there are no other nodes to receive the workloads. As a result, the Operator must be configured not to drain the workloads from the single node.
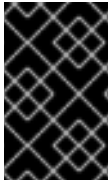
> **IMPORTANT**
>
> After performing the following procedure to disable draining workloads, you must remove any workload that uses an SR-IOV network interface before you change any SR-IOV network node policy.

Prerequisites

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

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

- You must have installed the SR-IOV Network Operator.

Procedure

- To set the **disableDrain** field to **true**, enter the following command:

  ```
  $ oc patch sriovoperatorconfig default --type=merge \
    -n openshift-sriov-network-operator \
    --patch '{ "spec": { "disableDrain": true } }'
  ```

> **TIP**
>
> You can alternatively apply the following YAML to update the Operator:
>
> ```
> apiVersion: sriovnetwork.openshift.io/v1
> kind: SriovOperatorConfig
> metadata:
>   name: default
>   namespace: openshift-sriov-network-operator
> spec:
>   disableDrain: true
> ```

26.3.1.9. Deploying the SR-IOV Operator for hosted control planes

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

After you configure and deploy your hosting service cluster, you can create a subscription to the SR-IOV Operator on a hosted cluster. The SR-IOV pod runs on worker machines rather than the control plane.

Prerequisites

You must configure and deploy the hosted cluster on AWS. For more information, see Configuring the hosting cluster on AWS (Technology Preview).

Procedure

1. Create a namespace and an Operator group:

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

2. Create a subscription to the SR-IOV Operator:

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: sriov-network-operator-subsription
  namespace: openshift-sriov-network-operator
spec:
  channel: "4.13"
  name: sriov-network-operator
  config:
    nodeSelector:
      node-role.kubernetes.io/worker: ""
  source: s/qe-app-registry/redhat-operators
  sourceNamespace: openshift-marketplace
```

Verification

1. To verify that the SR-IOV Operator is ready, run the following command and view the resulting output:

```
$ oc get csv -n openshift-sriov-network-operator
```

Example output

```
NAME                               DISPLAY               VERSION       REPLACES
PHASE
sriov-network-operator.4.13.0-202211021237   SR-IOV Network Operator   4.13.0-
202211021237   sriov-network-operator.4.13.0-202210290517   Succeeded
```

2. To verify that the SR-IOV pods are deployed, run the following command:

```
$ oc get pods -n openshift-sriov-network-operator
```

26.3.2. Next steps

- [Configuring an SR-IOV network device](#)

26.4. CONFIGURING AN SR-IOV NETWORK DEVICE

You can configure a Single Root I/O Virtualization (SR-IOV) device in your cluster.

26.4.1. SR-IOV network node configuration object

You specify the SR-IOV network device configuration for a node by creating an SR-IOV network node policy. The API object for the policy is part of the **sriovnetwork.openshift.io** API group.

The following YAML describes an SR-IOV network node policy:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: <name>                                           1
  namespace: openshift-sriov-network-operator             2
spec:
  resourceName: <sriov_resource_name>                     3
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"  4
  priority: <priority>                                    5
  mtu: <mtu>                                              6
  needVhostNet: false                                     7
  numVfs: <num>                                           8
  nicSelector:                                            9
    vendor: "<vendor_code>"                               10
    deviceID: "<device_id>"                               11
    pfNames: ["<pf_name>", ...]                           12
    rootDevices: ["<pci_bus_id>", ...]                    13
    netFilter: "<filter_string>"                          14
  deviceType: <device_type>                               15
  isRdma: false                                           16
  linkType: <link_type>                                   17
  eSwitchMode: <mode>                                     18
  excludeTopology: false                                  19
```

1   The name for the custom resource object.

2   The namespace where the SR-IOV Network Operator is installed.

3   The resource name of the SR-IOV network device plugin. You can create multiple SR-IOV network node policies for a resource name.

       When specifying a name, be sure to use the accepted syntax expression **^[a-zA-Z0-9_]+$** in the **resourceName**.

4   The node selector specifies the nodes to configure. Only SR-IOV network devices on the selected nodes are configured. The SR-IOV Container Network Interface (CNI) plugin and device plugin are deployed on selected nodes only.

> **IMPORTANT**
>
> The SR-IOV Network Operator applies node network configuration policies to nodes in sequence. Before applying node network configuration policies, the SR-IOV Network Operator checks if the machine config pool (MCP) for a node is in an unhealthy state such as **Degraded** or **Updating**. If a node is in an unhealthy MCP, the process of applying node network configuration policies to all targeted nodes in the cluster pauses until the MCP returns to a healthy state.
>
> To avoid a node in an unhealthy MCP from blocking the application of node network configuration policies to other nodes, including nodes in other MCPs, you must create a separate node network configuration policy for each MCP.

5 Optional: The priority is an integer value between **0** and **99**. A smaller value receives higher priority. For example, a priority of **10** is a higher priority than **99**. The default value is **99**.

6 Optional: The maximum transmission unit (MTU) of the virtual function. The maximum MTU value can vary for different network interface controller (NIC) models.

> **IMPORTANT**
>
> If you want to create virtual function on the default network interface, ensure that the MTU is set to a value that matches the cluster MTU.

7 Optional: Set **needVhostNet** to **true** to mount the **/dev/vhost-net** device in the pod. Use the mounted **/dev/vhost-net** device with Data Plane Development Kit (DPDK) to forward traffic to the kernel network stack.

8 The number of the virtual functions (VF) to create for the SR-IOV physical network device. For an Intel network interface controller (NIC), the number of VFs cannot be larger than the total VFs supported by the device. For a Mellanox NIC, the number of VFs cannot be larger than **128**.

9 The NIC selector identifies the device for the Operator to configure. You do not have to specify values for all the parameters. It is recommended to identify the network device with enough precision to avoid selecting a device unintentionally.

If you specify **rootDevices**, you must also specify a value for **vendor**, **deviceID**, or **pfNames**. If you specify both **pfNames** and **rootDevices** at the same time, ensure that they refer to the same device. If you specify a value for **netFilter**, then you do not need to specify any other parameter because a network ID is unique.

10 Optional: The vendor hexadecimal code of the SR-IOV network device. The only allowed values are **8086** and **15b3**.

11 Optional: The device hexadecimal code of the SR-IOV network device. For example, **101b** is the device ID for a Mellanox ConnectX-6 device.

12 Optional: An array of one or more physical function (PF) names for the device.

13 Optional: An array of one or more PCI bus addresses for the PF of the device. Provide the address in the following format: **0000:02:00.1**.

14 Optional: The platform-specific network filter. The only supported platform is Red Hat OpenStack Platform (RHOSP). Acceptable values use the following format: **openstack/NetworkID:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx**. Replace **xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx** with the value from the **/var/config/openstack/latest/network_data.json** metadata file.

15   Optional: The driver type for the virtual functions. The only allowed values are **netdevice** and **vfio-pci**. The default value is **netdevice**.

For a Mellanox NIC to work in DPDK mode on bare metal nodes, use the **netdevice** driver type and set **isRdma** to **true**.

16   Optional: Configures whether to enable remote direct memory access (RDMA) mode. The default value is **false**.

If the **isRdma** parameter is set to **true**, you can continue to use the RDMA-enabled VF as a normal network device. A device can be used in either mode.

Set **isRdma** to **true** and additionally set **needVhostNet** to **true** to configure a Mellanox NIC for use with Fast Datapath DPDK applications.

17   Optional: The link type for the VFs. The default value is **eth** for Ethernet. Change this value to 'ib' for InfiniBand.

When **linkType** is set to **ib**, **isRdma** is automatically set to **true** by the SR-IOV Network Operator webhook. When **linkType** is set to **ib**, **deviceType** should not be set to **vfio-pci**.

Do not set linkType to 'eth' for SriovNetworkNodePolicy, because this can lead to an incorrect number of available devices reported by the device plugin.

18   Optional: The NIC device mode. The only allowed values are **legacy** or **switchdev**.

When **eSwitchMode** is set to **legacy**, the default SR-IOV behavior is enabled.

When **eSwitchMode** is set to **switchdev**, hardware offloading is enabled.

19   Optional: To exclude advertising an SR-IOV network resource's NUMA node to the Topology Manager, set the value to **true**. The default value is **false**.

26.4.1.1. SR-IOV network node configuration examples

The following example describes the configuration for an InfiniBand device:

Example configuration for an InfiniBand device

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: policy-ib-net-1
  namespace: openshift-sriov-network-operator
spec:
  resourceName: ibnic1
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  numVfs: 4
  nicSelector:
    vendor: "15b3"
    deviceID: "101b"
    rootDevices:
```

```
    - "0000:19:00.0"
linkType: ib
isRdma: true
```

The following example describes the configuration for an SR-IOV network device in a RHOSP virtual machine:

Example configuration for an SR-IOV device in a virtual machine

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: policy-sriov-net-openstack-1
  namespace: openshift-sriov-network-operator
spec:
  resourceName: sriovnic1
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  numVfs: 1 1
  nicSelector:
    vendor: "15b3"
    deviceID: "101b"
    netFilter: "openstack/NetworkID:ea24bd04-8674-4f69-b0ee-fa0b3bd20509" 2
```

1   The **numVfs** field is always set to **1** when configuring the node network policy for a virtual machine.

2   The **netFilter** field must refer to a network ID when the virtual machine is deployed on RHOSP. Valid values for **netFilter** are available from an **SriovNetworkNodeState** object.

26.4.1.2. Virtual function (VF) partitioning for SR-IOV devices

In some cases, you might want to split virtual functions (VFs) from the same physical function (PF) into multiple resource pools. For example, you might want some of the VFs to load with the default driver and the remaining VFs load with the **vfio-pci** driver. In such a deployment, the **pfNames** selector in your SriovNetworkNodePolicy custom resource (CR) can be used to specify a range of VFs for a pool using the following format: **<pfname>#<first_vf>-<last_vf>**.

For example, the following YAML shows the selector for an interface named **netpf0** with VF **2** through **7**:

```
pfNames: ["netpf0#2-7"]
```

- **netpf0** is the PF interface name.

- **2** is the first VF index (0-based) that is included in the range.

- **7** is the last VF index (0-based) that is included in the range.

You can select VFs from the same PF by using different policy CRs if the following requirements are met:

- The **numVfs** value must be identical for policies that select the same PF.

- The VF index must be in the range of **0** to **<numVfs>-1**. For example, if you have a policy with **numVfs** set to **8**, then the **<first_vf>** value must not be smaller than **0**, and the **<last_vf>** must not be larger than **7**.

- The VFs ranges in different policies must not overlap.

- The **<first_vf>** must not be larger than the **<last_vf>**.

The following example illustrates NIC partitioning for an SR-IOV device.

The policy **policy-net-1** defines a resource pool **net-1** that contains the VF **0** of PF **netpf0** with the default VF driver. The policy **policy-net-1-dpdk** defines a resource pool **net-1-dpdk** that contains the VF **8** to **15** of PF **netpf0** with the **vfio** VF driver.

Policy **policy-net-1**:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: policy-net-1
  namespace: openshift-sriov-network-operator
spec:
  resourceName: net1
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  numVfs: 16
  nicSelector:
    pfNames: ["netpf0#0-0"]
  deviceType: netdevice
```

Policy **policy-net-1-dpdk**:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: policy-net-1-dpdk
  namespace: openshift-sriov-network-operator
spec:
  resourceName: net1dpdk
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  numVfs: 16
  nicSelector:
    pfNames: ["netpf0#8-15"]
  deviceType: vfio-pci
```

Verifying that the interface is successfully partitioned

Confirm that the interface partitioned to virtual functions (VFs) for the SR-IOV device by running the following command.

```
$ ip link show <interface>   ❶
```

❶ Replace **<interface>** with the interface that you specified when partitioning to VFs for the SR-IOV device, for example, **ens3f1**.

Example output

```
5: ens3f1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode
DEFAULT group default qlen 1000
link/ether 3c:fd:fe:d1:bc:01 brd ff:ff:ff:ff:ff:ff

vf 0     link/ether 5a:e7:88:25:ea:a0 brd ff:ff:ff:ff:ff:ff, spoof checking on, link-state auto, trust off
vf 1     link/ether 3e:1d:36:d7:3d:49 brd ff:ff:ff:ff:ff:ff, spoof checking on, link-state auto, trust off
vf 2     link/ether ce:09:56:97:df:f9 brd ff:ff:ff:ff:ff:ff, spoof checking on, link-state auto, trust off
vf 3     link/ether 5e:91:cf:88:d1:38 brd ff:ff:ff:ff:ff:ff, spoof checking on, link-state auto, trust off
vf 4     link/ether e6:06:a1:96:2f:de brd ff:ff:ff:ff:ff:ff, spoof checking on, link-state auto, trust off
```

26.4.2. Configuring SR-IOV network devices

The SR-IOV Network Operator adds the **SriovNetworkNodePolicy.sriovnetwork.openshift.io** CustomResourceDefinition to OpenShift Container Platform. You can configure an SR-IOV network device by creating a SriovNetworkNodePolicy custom resource (CR).
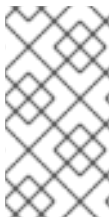
> **NOTE**
>
> When applying the configuration specified in a **SriovNetworkNodePolicy** object, the SR-IOV Operator might drain the nodes, and in some cases, reboot nodes.
>
> It might take several minutes for a configuration change to apply.

Prerequisites

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

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

- You have installed the SR-IOV Network Operator.

- You have enough available nodes in your cluster to handle the evicted workload from drained nodes.

- You have not selected any control plane nodes for SR-IOV network device configuration.

Procedure

1. Create an **SriovNetworkNodePolicy** object, and then save the YAML in the **<name>-sriov-node-network.yaml** file. Replace **<name>** with the name for this configuration.

2. Optional: Label the SR-IOV capable cluster nodes with **SriovNetworkNodePolicy.Spec.NodeSelector** if they are not already labeled. For more information about labeling nodes, see "Understanding how to update labels on nodes".

3. Create the **SriovNetworkNodePolicy** object:

   ```
   $ oc create -f <name>-sriov-node-network.yaml
   ```

   where **<name>** specifies the name for this configuration.

   After applying the configuration update, all the pods in **sriov-network-operator** namespace transition to the **Running** status.

4. To verify that the SR-IOV network device is configured, enter the following command. Replace **<node_name>** with the name of a node with the SR-IOV network device that you just configured.

```
$ oc get sriovnetworknodestates -n openshift-sriov-network-operator <node_name> -o
jsonpath='{.status.syncStatus}'
```

Additional resources

- [Understanding how to update labels on nodes](#) .

26.4.3. Troubleshooting SR-IOV configuration

After following the procedure to configure an SR-IOV network device, the following sections address some error conditions.

To display the state of nodes, run the following command:

```
$ oc get sriovnetworknodestates -n openshift-sriov-network-operator <node_name>
```

where: **<node_name>** specifies the name of a node with an SR-IOV network device.

Error output: Cannot allocate memory

```
"lastSyncError": "write /sys/bus/pci/devices/0000:3b:00.1/sriov_numvfs: cannot allocate memory"
```

When a node indicates that it cannot allocate memory, check the following items:

- Confirm that global SR-IOV settings are enabled in the BIOS for the node.

- Confirm that VT-d is enabled in the BIOS for the node.

26.4.4. Assigning an SR-IOV network to a VRF

As a cluster administrator, you can assign an SR-IOV network interface to your VRF domain by using the CNI VRF plugin.

To do this, add the VRF configuration to the optional **metaPlugins** parameter of the **SriovNetwork** resource.
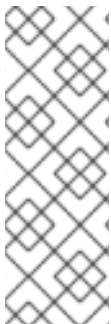
> **NOTE**
>
> Applications that use VRFs need to bind to a specific device. The common usage is to use the **SO_BINDTODEVICE** option for a socket. **SO_BINDTODEVICE** binds the socket to a device that is specified in the passed interface name, for example, **eth1**. To use **SO_BINDTODEVICE**, the application must have **CAP_NET_RAW** capabilities.
>
> Using a VRF through the **ip vrf exec** command is not supported in OpenShift Container Platform pods. To use VRF, bind applications directly to the VRF interface.

26.4.4.1. Creating an additional SR-IOV network attachment with the CNI VRF plugin

The SR-IOV Network Operator manages additional network definitions. When you specify an additional SR-IOV network to create, the SR-IOV Network Operator creates the **NetworkAttachmentDefinition** custom resource (CR) automatically.

> **NOTE**
>
> Do not edit **NetworkAttachmentDefinition** custom resources that the SR-IOV Network Operator manages. Doing so might disrupt network traffic on your additional network.

To create an additional SR-IOV network attachment with the CNI VRF plugin, perform the following procedure.

Prerequisites

- Install the OpenShift Container Platform CLI (oc).

- Log in to the OpenShift Container Platform cluster as a user with cluster-admin privileges.

Procedure

1. Create the **SriovNetwork** custom resource (CR) for the additional SR-IOV network attachment and insert the **metaPlugins** configuration, as in the following example CR. Save the YAML as the file **sriov-network-attachment.yaml**.

   ```
   apiVersion: sriovnetwork.openshift.io/v1
   kind: SriovNetwork
   metadata:
     name: example-network
     namespace: additional-sriov-network-1
   spec:
     ipam: |
       {
         "type": "host-local",
         "subnet": "10.56.217.0/24",
         "rangeStart": "10.56.217.171",
         "rangeEnd": "10.56.217.181",
         "routes": [{
           "dst": "0.0.0.0/0"
         }],
         "gateway": "10.56.217.1"
       }
     vlan: 0
     resourceName: intelnics
     metaPlugins : |
       {
         "type": "vrf",           1
         "vrfname": "example-vrf-name"   2
       }
   ```

   1 **type** must be set to **vrf**.

   2 **vrfname** is the name of the VRF that the interface is assigned to. If it does not exist in the pod, it is created.

2. Create the **SriovNetwork** resource:

```
$ oc create -f sriov-network-attachment.yaml
```

Verifying that the **NetworkAttachmentDefinition** CR is successfully created

- Confirm that the SR-IOV Network Operator created the **NetworkAttachmentDefinition** CR by running the following command.

```
$ oc get network-attachment-definitions -n <namespace> ❶
```

❶   Replace **<namespace>** with the namespace that you specified when configuring the network attachment, for example, **additional-sriov-network-1**.

Example output

```
NAME                     AGE
additional-sriov-network-1     14m
```
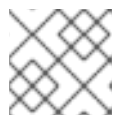
> **NOTE**
>
> There might be a delay before the SR-IOV Network Operator creates the CR.

Verifying that the additional SR-IOV network attachment is successful

To verify that the VRF CNI is correctly configured and the additional SR-IOV network attachment is attached, do the following:

1. Create an SR-IOV network that uses the VRF CNI.

2. Assign the network to a pod.

3. Verify that the pod network attachment is connected to the SR-IOV additional network. Remote shell into the pod and run the following command:

```
$ ip vrf show
```

Example output

```
Name          Table
-----------------------
red           10
```

4. Confirm the VRF interface is master of the secondary interface:

```
$ ip link
```

Example output

```
...
5: net1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master red
```

> state UP mode
> ...

26.4.5. Exclude the SR-IOV network topology for NUMA-aware scheduling

You can exclude advertising the Non-Uniform Memory Access (NUMA) node for the SR-IOV network to the Topology Manager for more flexible SR-IOV network deployments during NUMA-aware pod scheduling.

In some scenarios, it is a priority to maximize CPU and memory resources for a pod on a single NUMA node. By not providing a hint to the Topology Manager about the NUMA node for the pod's SR-IOV network resource, the Topology Manager can deploy the SR-IOV network resource and the pod CPU and memory resources to different NUMA nodes. This can add to network latency because of the data transfer between NUMA nodes. However, it is acceptable in scenarios when workloads require optimal CPU and memory performance.

For example, consider a compute node, **compute-1**, that features two NUMA nodes: **numa0** and **numa1**. The SR-IOV-enabled NIC is present on **numa0**. The CPUs available for pod scheduling are present on **numa1** only. By setting the **excludeTopology** specification to **true**, the Topology Manager can assign CPU and memory resources for the pod to **numa1** and can assign the SR-IOV network resource for the same pod to **numa0**. This is only possible when you set the **excludeTopology** specification to **true**. Otherwise, the Topology Manager attempts to place all resources on the same NUMA node.

26.4.5.1. Excluding the SR-IOV network topology for NUMA-aware scheduling

To exclude advertising the SR-IOV network resource's Non-Uniform Memory Access (NUMA) node to the Topology Manager, you can configure the **excludeTopology** specification in the **SriovNetworkNodePolicy** custom resource. Use this configuration for more flexible SR-IOV network deployments during NUMA-aware pod scheduling.

Prerequisites

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

- You have configured the CPU Manager policy to **static**. For more information about CPU Manager, see the *Additional resources* section.

- You have configured the Topology Manager policy to **single-numa-node**.

- You have installed the SR-IOV Network Operator.

Procedure

1. Create the **SriovNetworkNodePolicy** CR:

   a. Save the following YAML in the **sriov-network-node-policy.yaml** file, replacing values in the YAML to match your environment:

      ```
      apiVersion: sriovnetwork.openshift.io/v1
      kind: SriovNetworkNodePolicy
      metadata:
        name: <policy_name>
        namespace: openshift-sriov-network-operator
      spec:
        resourceName: sriovnuma0 1
      ```

```
    nodeSelector:
      kubernetes.io/hostname: <node_name>
    numVfs: <number_of_Vfs>
    nicSelector: 2
      vendor: "<vendor_ID>"
      deviceID: "<device_ID>"
    deviceType: netdevice
    excludeTopology: true 3
```

1 The resource name of the SR-IOV network device plugin. This YAML uses a sample **resourceName** value.

2 Identify the device for the Operator to configure by using the NIC selector.

3 To exclude advertising the NUMA node for the SR-IOV network resource to the Topology Manager, set the value to **true**. The default value is **false**.

> **NOTE**
>
> If multiple **SriovNetworkNodePolicy** resources target the same SR-IOV network resource, the **SriovNetworkNodePolicy** resources must have the same value as the **excludeTopology** specification. Otherwise, the conflicting policy is rejected.

b. Create the **SriovNetworkNodePolicy** resource by running the following command:

```
$ oc create -f sriov-network-node-policy.yaml
```

Example output

```
sriovnetworknodepolicy.sriovnetwork.openshift.io/policy-for-numa-0 created
```

2. Create the **SriovNetwork** CR:

a. Save the following YAML in the **sriov-network.yaml** file, replacing values in the YAML to match your environment:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: sriov-numa-0-network 1
  namespace: openshift-sriov-network-operator
spec:
  resourceName: sriovnuma0 2
  networkNamespace: <namespace> 3
  ipam: |- 4
    {
      "type": "<ipam_type>",
    }
```

1 Replace **sriov-numa-0-network** with the name for the SR-IOV network resource.

2 Specify the resource name for the **SriovNetworkNodePolicy** CR from the previous step. This YAML uses a sample **resourceName** value.

3 Enter the namespace for your SR-IOV network resource.

4 Enter the IP address management configuration for the SR-IOV network.

b. Create the **SriovNetwork** resource by running the following command:

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

Example output

```
sriovnetwork.sriovnetwork.openshift.io/sriov-numa-0-network created
```

3. Create a pod and assign the SR-IOV network resource from the previous step:

a. Save the following YAML in the **sriov-network-pod.yaml** file, replacing values in the YAML to match your environment:

```
apiVersion: v1
kind: Pod
metadata:
  name: <pod_name>
  annotations:
    k8s.v1.cni.cncf.io/networks: |-
      [
        {
          "name": "sriov-numa-0-network", 1
        }
      ]
spec:
  containers:
  - name: <container_name>
    image: <image>
    imagePullPolicy: IfNotPresent
    command: ["sleep", "infinity"]
```

1 This is the name of the **SriovNetwork** resource that uses the **SriovNetworkNodePolicy** resource.

b. Create the **Pod** resource by running the following command:

```
$ oc create -f sriov-network-pod.yaml
```

Example output

```
pod/example-pod created
```

Verification

1. Verify the status of the pod by running the following command, replacing **<pod_name>** with the name of the pod:

```
$ oc get pod <pod_name>
```

Example output

```
NAME                                READY   STATUS    RESTARTS   AGE
test-deployment-sriov-76cbbf4756-k9v72   1/1     Running   0          45h
```

2. Open a debug session with the target pod to verify that the SR-IOV network resources are deployed to a different node than the memory and CPU resources.

   a. Open a debug session with the pod by running the following command, replacing <pod_name> with the target pod name.

   ```
   $ oc debug pod/<pod_name>
   ```

   b. Set **/host** as the root directory within the debug shell. The debug pod mounts the root file system from the host in **/host** within the pod. By changing the root directory to **/host**, you can run binaries from the host file system:

   ```
   $ chroot /host
   ```

   c. View information about the CPU allocation by running the following commands:

   ```
   $ lscpu | grep NUMA
   ```

   Example output

   ```
   NUMA node(s):           2
   NUMA node0 CPU(s):      0,2,4,6,8,10,12,14,16,18,...
   NUMA node1 CPU(s):      1,3,5,7,9,11,13,15,17,19,...
   ```

   ```
   $ cat /proc/self/status | grep Cpus
   ```

   Example output

   ```
   Cpus_allowed: aa
   Cpus_allowed_list: 1,3,5,7
   ```

   ```
   $ cat /sys/class/net/net1/device/numa_node
   ```

   Example output

   ```
   0
   ```

   In this example, CPUs 1,3,5, and 7 are allocated to **NUMA node1** but the SR-IOV network resource can use the NIC in **NUMA node0**.
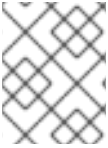
	NOTE
	If the **excludeTopology** specification is set to **True**, it is possible that the required resources exist in the same NUMA node.

Additional resources

- [Using CPU Manager](#)

26.4.6. Next steps

- [Configuring an SR-IOV network attachment](#)

26.5. CONFIGURING AN SR-IOV ETHERNET NETWORK ATTACHMENT

You can configure an Ethernet network attachment for an Single Root I/O Virtualization (SR-IOV) device in the cluster.

26.5.1. Ethernet device configuration object

You can configure an Ethernet network device by defining an **SriovNetwork** object.

The following YAML describes an **SriovNetwork** object:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: <name> 1
  namespace: openshift-sriov-network-operator 2
spec:
  resourceName: <sriov_resource_name> 3
  networkNamespace: <target_namespace> 4
  vlan: <vlan> 5
  spoofChk: "<spoof_check>" 6
  ipam: |- 7
    {}
  linkState: <link_state> 8
  maxTxRate: <max_tx_rate> 9
  minTxRate: <min_tx_rate> 10
  vlanQoS: <vlan_qos> 11
  trust: "<trust_vf>" 12
  capabilities: <capabilities> 13
```

1 A name for the object. The SR-IOV Network Operator creates a **NetworkAttachmentDefinition** object with same name.

2 The namespace where the SR-IOV Network Operator is installed.

3 The value for the **spec.resourceName** parameter from the **SriovNetworkNodePolicy** object that defines the SR-IOV hardware for this additional network.

4 The target namespace for the **SriovNetwork** object. Only pods in the target namespace can attach to the additional network.

5 Optional: A Virtual LAN (VLAN) ID for the additional network. The integer value must be from **0** to **4095**. The default value is **0**.

6 Optional: The spoof check mode of the VF. The allowed values are the strings **"on"** and **"off"**.

> **IMPORTANT**
>
> You must enclose the value you specify in quotes or the object is rejected by the SR-IOV Network Operator.

7 A configuration object for the IPAM CNI plugin as a YAML block scalar. The plugin manages IP address assignment for the attachment definition.

8 Optional: The link state of virtual function (VF). Allowed value are **enable**, **disable** and **auto**.

9 Optional: A maximum transmission rate, in Mbps, for the VF.

10 Optional: A minimum transmission rate, in Mbps, for the VF. This value must be less than or equal to the maximum transmission rate.

> **NOTE**
>
> Intel NICs do not support the **minTxRate** parameter. For more information, see BZ#1772847.

11 Optional: An IEEE 802.1p priority level for the VF. The default value is **0**.

12 Optional: The trust mode of the VF. The allowed values are the strings **"on"** and **"off"**.

> **IMPORTANT**
>
> You must enclose the value that you specify in quotes, or the SR-IOV Network Operator rejects the object.

13 Optional: The capabilities to configure for this additional network. You can specify **"{ "ips": true }"** to enable IP address support or **"{ "mac": true }"** to enable MAC address support.

26.5.1.1. Configuration of IP address assignment for an additional network

The IP address management (IPAM) Container Network Interface (CNI) plugin provides IP addresses for other CNI plugins.

You can use the following IP address assignment types:

- Static assignment.

- Dynamic assignment through a DHCP server. The DHCP server you specify must be reachable from the additional network.

- Dynamic assignment through the Whereabouts IPAM CNI plugin.

26.5.1.1.1. Static IP address assignment configuration

The following table describes the configuration for static IP address assignment:

Table 26.3. **ipam** static configuration object

Field	Type	Description
type	**string**	The IPAM address type. The value **static** is required.
addresses	**array**	An array of objects specifying IP addresses to assign to the virtual interface. Both IPv4 and IPv6 IP addresses are supported.
routes	**array**	An array of objects specifying routes to configure inside the pod.
dns	**array**	Optional: An array of objects specifying the DNS configuration.

The **addresses** array requires objects with the following fields:

Table 26.4. **ipam.addresses[]** array

Field	Type	Description
address	**string**	An IP address and network prefix that you specify. For example, if you specify **10.10.21.10/24**, then the additional network is assigned an IP address of **10.10.21.10** and the netmask is **255.255.255.0**.
gateway	**string**	The default gateway to route egress network traffic to.

Table 26.5. **ipam.routes[]** array

Field	Type	Description
dst	**string**	The IP address range in CIDR format, such as **192.168.17.0/24** or **0.0.0.0/0** for the default route.
gw	**string**	The gateway where network traffic is routed.

Table 26.6. **ipam.dns** object

Field	Type	Description
nameservers	**array**	An array of one or more IP addresses for to send DNS queries to.
domain	**array**	The default domain to append to a hostname. For example, if the domain is set to **example.com**, a DNS lookup query for **example-host** is rewritten as **example-host.example.com**.

Field	Type	Description
search	**array**	An array of domain names to append to an unqualified hostname, such as **example-host**, during a DNS lookup query.

Static IP address assignment configuration example

```
{
  "ipam": {
    "type": "static",
     "addresses": [
       {
         "address": "191.168.1.7/24"
       }
     ]
  }
}
```

26.5.1.1.2. Dynamic IP address (DHCP) assignment configuration

The following JSON describes the configuration for dynamic IP address address assignment with DHCP.

RENEWAL OF DHCP LEASES

A pod obtains its original DHCP lease when it is created. The lease must be periodically renewed by a minimal DHCP server deployment running on the cluster.

The SR-IOV Network Operator does not create a DHCP server deployment; The Cluster Network Operator is responsible for creating the minimal DHCP server deployment.

To trigger the deployment of the DHCP server, you must create a shim network attachment by editing the Cluster Network Operator configuration, as in the following example:

Example shim network attachment definition

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  additionalNetworks:
  - name: dhcp-shim
    namespace: default
    type: Raw
    rawCNIConfig: |-
      {
        "name": "dhcp-shim",
        "cniVersion": "0.3.1",
        "type": "bridge",
        "ipam": {
          "type": "dhcp"
        }
      }
  # ...
```

Table 26.7. **ipam** DHCP configuration object

Field	Type	Description
type	**string**	The IPAM address type. The value **dhcp** is required.

Dynamic IP address (DHCP) assignment configuration example

```
{
  "ipam": {
    "type": "dhcp"
  }
}
```

26.5.1.1.3. Dynamic IP address assignment configuration with Whereabouts

The Whereabouts CNI plugin allows the dynamic assignment of an IP address to an additional network without the use of a DHCP server.

The following table describes the configuration for dynamic IP address assignment with Whereabouts:

Table 26.8. **ipam** whereabouts configuration object

Field	Type	Description
type	**string**	The IPAM address type. The value **whereabouts** is required.
range	**string**	An IP address and range in CIDR notation. IP addresses are assigned from within this range of addresses.
exclude	**array**	Optional: A list of zero or more IP addresses and ranges in CIDR notation. IP addresses within an excluded address range are not assigned.

Dynamic IP address assignment configuration example that uses Whereabouts

```
{
  "ipam": {
    "type": "whereabouts",
    "range": "192.0.2.192/27",
    "exclude": [
       "192.0.2.192/30",
       "192.0.2.196/32"
    ]
  }
}
```

26.5.2. Configuring SR-IOV additional network

You can configure an additional network that uses SR-IOV hardware by creating an **SriovNetwork** object. When you create an **SriovNetwork** object, the SR-IOV Network Operator automatically creates a **NetworkAttachmentDefinition** object.
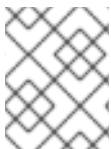
> **NOTE**
>
> Do not modify or delete an **SriovNetwork** object if it is attached to any pods in a **running** state.

Prerequisites

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

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

Procedure

1. Create a **SriovNetwork** object, and then save the YAML in the **<name>.yaml** file, where **<name>** is a name for this additional network. The object specification might resemble the following example:

   ```
   apiVersion: sriovnetwork.openshift.io/v1
   ```

```
kind: SriovNetwork
metadata:
  name: attach1
  namespace: openshift-sriov-network-operator
spec:
  resourceName: net1
  networkNamespace: project2
  ipam: |-
    {
      "type": "host-local",
      "subnet": "10.56.217.0/24",
      "rangeStart": "10.56.217.171",
      "rangeEnd": "10.56.217.181",
      "gateway": "10.56.217.1"
    }
```

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

   ```
   $ oc create -f <name>.yaml
   ```

   where **<name>** specifies the name of the additional network.

3. Optional: To confirm that the **NetworkAttachmentDefinition** object that is associated with the **SriovNetwork** object that you created in the previous step exists, enter the following command. Replace **<namespace>** with the networkNamespace you specified in the **SriovNetwork** object.

   ```
   $ oc get net-attach-def -n <namespace>
   ```

26.5.3. Next steps

- [Adding a pod to an SR-IOV additional network](#)

26.5.4. Additional resources

- [Configuring an SR-IOV network device](#)

26.6. CONFIGURING AN SR-IOV INFINIBAND NETWORK ATTACHMENT

You can configure an InfiniBand (IB) network attachment for an Single Root I/O Virtualization (SR-IOV) device in the cluster.

26.6.1. InfiniBand device configuration object

You can configure an InfiniBand (IB) network device by defining an **SriovIBNetwork** object.

The following YAML describes an **SriovIBNetwork** object:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovIBNetwork
metadata:
  name: <name>      1
  namespace: openshift-sriov-network-operator      2
```

```
    spec:
      resourceName: <sriov_resource_name> 3
      networkNamespace: <target_namespace> 4
      ipam: |- 5
        {}
      linkState: <link_state> 6
      capabilities: <capabilities> 7
```

1   A name for the object. The SR-IOV Network Operator creates a **NetworkAttachmentDefinition** object with same name.

2   The namespace where the SR-IOV Operator is installed.

3   The value for the **spec.resourceName** parameter from the **SriovNetworkNodePolicy** object that defines the SR-IOV hardware for this additional network.

4   The target namespace for the **SriovIBNetwork** object. Only pods in the target namespace can attach to the network device.

5   Optional: A configuration object for the IPAM CNI plugin as a YAML block scalar. The plugin manages IP address assignment for the attachment definition.

6   Optional: The link state of virtual function (VF). Allowed values are **enable**, **disable** and **auto**.

7   Optional: The capabilities to configure for this network. You can specify **"{ "ips": true }"** to enable IP address support or **"{ "infinibandGUID": true }"** to enable IB Global Unique Identifier (GUID) support.

26.6.1.1. Configuration of IP address assignment for an additional network

The IP address management (IPAM) Container Network Interface (CNI) plugin provides IP addresses for other CNI plugins.

You can use the following IP address assignment types:

- Static assignment.

- Dynamic assignment through a DHCP server. The DHCP server you specify must be reachable from the additional network.

- Dynamic assignment through the Whereabouts IPAM CNI plugin.

26.6.1.1.1. Static IP address assignment configuration

The following table describes the configuration for static IP address assignment:

Table 26.9. **ipam** static configuration object

Field	Type	Description
type	**string**	The IPAM address type. The value **static** is required.

Field	Type	Description
addresses	**array**	An array of objects specifying IP addresses to assign to the virtual interface. Both IPv4 and IPv6 IP addresses are supported.
routes	**array**	An array of objects specifying routes to configure inside the pod.
dns	**array**	Optional: An array of objects specifying the DNS configuration.

The **addresses** array requires objects with the following fields:

Table 26.10. **ipam.addresses[]** array

Field	Type	Description
address	**string**	An IP address and network prefix that you specify. For example, if you specify **10.10.21.10/24**, then the additional network is assigned an IP address of **10.10.21.10** and the netmask is **255.255.255.0**.
gateway	**string**	The default gateway to route egress network traffic to.

Table 26.11. **ipam.routes[]** array

Field	Type	Description
dst	**string**	The IP address range in CIDR format, such as **192.168.17.0/24** or **0.0.0.0/0** for the default route.
gw	**string**	The gateway where network traffic is routed.

Table 26.12. **ipam.dns** object

Field	Type	Description
nameservers	**array**	An array of one or more IP addresses for to send DNS queries to.
domain	**array**	The default domain to append to a hostname. For example, if the domain is set to **example.com**, a DNS lookup query for **example-host** is rewritten as **example-host.example.com**.
search	**array**	An array of domain names to append to an unqualified hostname, such as **example-host**, during a DNS lookup query.

Static IP address assignment configuration example

```
{
```

```
    "ipam": {
     "type": "static",
      "addresses": [
        {
          "address": "191.168.1.7/24"
        }
      ]
    }
  }
```

26.6.1.1.2. Dynamic IP address (DHCP) assignment configuration

The following JSON describes the configuration for dynamic IP address address assignment with DHCP.

> **RENEWAL OF DHCP LEASES**
>
> A pod obtains its original DHCP lease when it is created. The lease must be periodically renewed by a minimal DHCP server deployment running on the cluster.
>
> To trigger the deployment of the DHCP server, you must create a shim network attachment by editing the Cluster Network Operator configuration, as in the following example:
>
> **Example shim network attachment definition**
>
> ```yaml
> apiVersion: operator.openshift.io/v1
> kind: Network
> metadata:
>   name: cluster
> spec:
>   additionalNetworks:
>   - name: dhcp-shim
>     namespace: default
>     type: Raw
>     rawCNIConfig: |-
>       {
>         "name": "dhcp-shim",
>         "cniVersion": "0.3.1",
>         "type": "bridge",
>         "ipam": {
>           "type": "dhcp"
>         }
>       }
>     # ...
> ```

Table 26.13. **ipam** DHCP configuration object

Field	Type	Description
type	**string**	The IPAM address type. The value **dhcp** is required.

Dynamic IP address (DHCP) assignment configuration example

```
{
  "ipam": {
    "type": "dhcp"
  }
}
```

26.6.1.1.3. Dynamic IP address assignment configuration with Whereabouts

The Whereabouts CNI plugin allows the dynamic assignment of an IP address to an additional network without the use of a DHCP server.

The following table describes the configuration for dynamic IP address assignment with Whereabouts:

Table 26.14. **ipam** whereabouts configuration object

Field	Type	Description
type	**string**	The IPAM address type. The value **whereabouts** is required.
range	**string**	An IP address and range in CIDR notation. IP addresses are assigned from within this range of addresses.
exclude	**array**	Optional: A list of zero or more IP addresses and ranges in CIDR notation. IP addresses within an excluded address range are not assigned.

Dynamic IP address assignment configuration example that uses Whereabouts

```
{
  "ipam": {
    "type": "whereabouts",
    "range": "192.0.2.192/27",
    "exclude": [
       "192.0.2.192/30",
       "192.0.2.196/32"
    ]
  }
}
```

26.6.2. Configuring SR-IOV additional network

You can configure an additional network that uses SR-IOV hardware by creating an **SriovIBNetwork** object. When you create an **SriovIBNetwork** object, the SR-IOV Network Operator automatically creates a **NetworkAttachmentDefinition** object.

> **NOTE**
>
> Do not modify or delete an **SriovIBNetwork** object if it is attached to any pods in a **running** state.

Prerequisites

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

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

Procedure

1. Create a **SriovIBNetwork** object, and then save the YAML in the **<name>.yaml** file, where **<name>** is a name for this additional network. The object specification might resemble the following example:

   ```
   apiVersion: sriovnetwork.openshift.io/v1
   kind: SriovIBNetwork
   metadata:
     name: attach1
     namespace: openshift-sriov-network-operator
   spec:
     resourceName: net1
     networkNamespace: project2
     ipam: |-
       {
         "type": "host-local",
         "subnet": "10.56.217.0/24",
         "rangeStart": "10.56.217.171",
         "rangeEnd": "10.56.217.181",
         "gateway": "10.56.217.1"
       }
   ```

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

   ```
   $ oc create -f <name>.yaml
   ```

   where **<name>** specifies the name of the additional network.

3. Optional: To confirm that the **NetworkAttachmentDefinition** object that is associated with the **SriovIBNetwork** object that you created in the previous step exists, enter the following command. Replace **<namespace>** with the networkNamespace you specified in the **SriovIBNetwork** object.

   ```
   $ oc get net-attach-def -n <namespace>
   ```

26.6.3. Next steps

- [Adding a pod to an SR-IOV additional network](#)

26.6.4. Additional resources

- [Configuring an SR-IOV network device](#)

26.7. ADDING A POD TO AN SR-IOV ADDITIONAL NETWORK

You can add a pod to an existing Single Root I/O Virtualization (SR-IOV) network.

26.7.1. Runtime configuration for a network attachment

When attaching a pod to an additional network, you can specify a runtime configuration to make specific customizations for the pod. For example, you can request a specific MAC hardware address.

You specify the runtime configuration by setting an annotation in the pod specification. The annotation key is **k8s.v1.cni.cncf.io/networks**, and it accepts a JSON object that describes the runtime configuration.

26.7.1.1. Runtime configuration for an Ethernet-based SR-IOV attachment

The following JSON describes the runtime configuration options for an Ethernet-based SR-IOV network attachment.

```
[
  {
    "name": "<name>",          1
    "mac": "<mac_address>",    2
    "ips": ["<cidr_range>"]    3
  }
]
```

1. The name of the SR-IOV network attachment definition CR.

2. Optional: The MAC address for the SR-IOV device that is allocated from the resource type defined in the SR-IOV network attachment definition CR. To use this feature, you also must specify **{ "mac": true }** in the **SriovNetwork** object.

3. Optional: IP addresses for the SR-IOV device that is allocated from the resource type defined in the SR-IOV network attachment definition CR. Both IPv4 and IPv6 addresses are supported. To use this feature, you also must specify **{ "ips": true }** in the **SriovNetwork** object.

Example runtime configuration

```
apiVersion: v1
kind: Pod
metadata:
  name: sample-pod
  annotations:
    k8s.v1.cni.cncf.io/networks: |-
      [
        {
          "name": "net1",
          "mac": "20:04:0f:f1:88:01",
          "ips": ["192.168.10.1/24", "2001::1/64"]
        }
      ]
spec:
  containers:
  - name: sample-container
    image: <image>
    imagePullPolicy: IfNotPresent
    command: ["sleep", "infinity"]
```

26.7.1.2. Runtime configuration for an InfiniBand-based SR-IOV attachment

The following JSON describes the runtime configuration options for an InfiniBand-based SR-IOV network attachment.

```
[
  {
    "name": "<network_attachment>", ❶
    "infiniband-guid": "<guid>", ❷
    "ips": ["<cidr_range>"] ❸
  }
]
```

❶ The name of the SR-IOV network attachment definition CR.

❷ The InfiniBand GUID for the SR-IOV device. To use this feature, you also must specify **{ "infinibandGUID": true }** in the **SriovIBNetwork** object.

❸ The IP addresses for the SR-IOV device that is allocated from the resource type defined in the SR-IOV network attachment definition CR. Both IPv4 and IPv6 addresses are supported. To use this feature, you also must specify **{ "ips": true }** in the **SriovIBNetwork** object.

Example runtime configuration

```
apiVersion: v1
kind: Pod
metadata:
  name: sample-pod
  annotations:
    k8s.v1.cni.cncf.io/networks: |-
      [
        {
          "name": "ib1",
          "infiniband-guid": "c2:11:22:33:44:55:66:77",
          "ips": ["192.168.10.1/24", "2001::1/64"]
        }
      ]
spec:
  containers:
  - name: sample-container
    image: <image>
    imagePullPolicy: IfNotPresent
    command: ["sleep", "infinity"]
```

26.7.2. Adding a pod to an additional network

You can add a pod to an additional network. The pod continues to send normal cluster-related network traffic over the default network.

When a pod is created additional networks are attached to it. However, if a pod already exists, you cannot attach additional networks to it.

The pod must be in the same namespace as the additional network.

> **NOTE**
>
> The SR-IOV Network Resource Injector adds the **resource** field to the first container in a pod automatically.
>
> If you are using an Intel network interface controller (NIC) in Data Plane Development Kit (DPDK) mode, only the first container in your pod is configured to access the NIC. Your SR-IOV additional network is configured for DPDK mode if the **deviceType** is set to **vfio-pci** in the **SriovNetworkNodePolicy** object.
>
> You can work around this issue by either ensuring that the container that needs access to the NIC is the first container defined in the **Pod** object or by disabling the Network Resource Injector. For more information, see BZ#1990953.

Prerequisites

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

- Log in to the cluster.

- Install the SR-IOV Operator.

- Create either an **SriovNetwork** object or an **SriovIBNetwork** object to attach the pod to.

Procedure

1. Add an annotation to the **Pod** object. Only one of the following annotation formats can be used:

   a. To attach an additional network without any customization, add an annotation with the following format. Replace **<network>** with the name of the additional network to associate with the pod:

   ```
   metadata:
     annotations:
       k8s.v1.cni.cncf.io/networks: <network>[,<network>,...] ❶
   ```

   ❶ To specify more than one additional network, separate each network with a comma. Do not include whitespace between the comma. If you specify the same additional network multiple times, that pod will have multiple network interfaces attached to that network.

   b. To attach an additional network with customizations, add an annotation with the following format:

   ```
   metadata:
     annotations:
       k8s.v1.cni.cncf.io/networks: |-
         [
           {
             "name": "<network>", ❶
             "namespace": "<namespace>", ❷
             "default-route": ["<default-route>"] ❸
           }
         ]
   ```

1 Specify the name of the additional network defined by a
NetworkAttachmentDefinition object.

2 Specify the namespace where the **NetworkAttachmentDefinition** object is defined.

3 Optional: Specify an override for the default route, such as **192.168.17.1**.

2. To create the pod, enter the following command. Replace **<name>** with the name of the pod.

   ```
   $ oc create -f <name>.yaml
   ```

3. Optional: To Confirm that the annotation exists in the **Pod** CR, enter the following command,
   replacing **<name>** with the name of the pod.

   ```
   $ oc get pod <name> -o yaml
   ```

   In the following example, the **example-pod** pod is attached to the **net1** additional network:

   ```
   $ oc get pod example-pod -o yaml
   apiVersion: v1
   kind: Pod
   metadata:
     annotations:
       k8s.v1.cni.cncf.io/networks: macvlan-bridge
       k8s.v1.cni.cncf.io/network-status: |-  1
         [{
             "name": "openshift-sdn",
             "interface": "eth0",
             "ips": [
                 "10.128.2.14"
             ],
             "default": true,
             "dns": {}
         },{
             "name": "macvlan-bridge",
             "interface": "net1",
             "ips": [
                 "20.2.2.100"
             ],
             "mac": "22:2f:60:a5:f8:00",
             "dns": {}
         }]
     name: example-pod
     namespace: default
   spec:
     ...
   status:
     ...
   ```

   1 The **k8s.v1.cni.cncf.io/network-status** parameter is a JSON array of objects. Each object
   describes the status of an additional network attached to the pod. The annotation value is
   stored as a plain text value.

26.7.3. Creating a non-uniform memory access (NUMA) aligned SR-IOV pod

You can create a NUMA aligned SR-IOV pod by restricting SR-IOV and the CPU resources allocated from the same NUMA node with **restricted** or **single-numa-node** Topology Manager polices.

Prerequisites

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

- You have configured the CPU Manager policy to **static**. For more information on CPU Manager, see the "Additional resources" section.

- You have configured the Topology Manager policy to **single-numa-node**.

> **NOTE**
>
> When **single-numa-node** is unable to satisfy the request, you can configure the Topology Manager policy to **restricted**. For more flexible SR-IOV network resource scheduling, see *Excluding SR-IOV network topology during NUMA-aware scheduling* in the *Additional resources* section.

Procedure

1. Create the following SR-IOV pod spec, and then save the YAML in the **<name>-sriov-pod.yaml** file. Replace **<name>** with a name for this pod.
   The following example shows an SR-IOV pod spec:

   ```
   apiVersion: v1
   kind: Pod
   metadata:
     name: sample-pod
     annotations:
       k8s.v1.cni.cncf.io/networks: <name>  1
   spec:
     containers:
     - name: sample-container
       image: <image>  2
       command: ["sleep", "infinity"]
       resources:
         limits:
           memory: "1Gi"  3
           cpu: "2"  4
         requests:
           memory: "1Gi"
           cpu: "2"
   ```

   1 Replace **<name>** with the name of the SR-IOV network attachment definition CR.

   2 Replace **<image>** with the name of the **sample-pod** image.

   3 To create the SR-IOV pod with guaranteed QoS, set **memory limits** equal to **memory requests**.

   4 To create the SR-IOV pod with guaranteed QoS, set **cpu limits** equals to **cpu requests**.

2. Create the sample SR-IOV pod by running the following command:

```
$ oc create -f <filename> ❶
```

❶ Replace **<filename>** with the name of the file you created in the previous step.

3. Confirm that the **sample-pod** is configured with guaranteed QoS.

```
$ oc describe pod sample-pod
```

4. Confirm that the **sample-pod** is allocated with exclusive CPUs.

```
$ oc exec sample-pod -- cat /sys/fs/cgroup/cpuset/cpuset.cpus
```

5. Confirm that the SR-IOV device and CPUs that are allocated for the **sample-pod** are on the same NUMA node.

```
$ oc exec sample-pod -- cat /sys/fs/cgroup/cpuset/cpuset.cpus
```

26.7.4. A test pod template for clusters that use SR-IOV on OpenStack

The following **testpmd** pod demonstrates container creation with huge pages, reserved CPUs, and the SR-IOV port.

An example testpmd pod

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: testpmd-sriov
  namespace: mynamespace
  annotations:
    cpu-load-balancing.crio.io: "disable"
    cpu-quota.crio.io: "disable"
# ...
spec:
  containers:
  - name: testpmd
    command: ["sleep", "99999"]
    image: registry.redhat.io/openshift4/dpdk-base-rhel8:v4.9
    securityContext:
      capabilities:
        add: ["IPC_LOCK","SYS_ADMIN"]
      privileged: true
      runAsUser: 0
    resources:
      requests:
        memory: 1000Mi
        hugepages-1Gi: 1Gi
        cpu: '2'
        openshift.io/sriov1: 1
      limits:
        hugepages-1Gi: 1Gi
```

```
      cpu: '2'
      memory: 1000Mi
      openshift.io/sriov1: 1
  volumeMounts:
    - mountPath: /dev/hugepages
      name: hugepage
      readOnly: False
runtimeClassName: performance-cnf-performanceprofile 1
volumes:
- name: hugepage
  emptyDir:
    medium: HugePages
```

[1] This example assumes that the name of the performance profile is **cnf-performance profile**.

26.7.5. Additional resources

- [Configuring an SR-IOV Ethernet network attachment](#)

- [Configuring an SR-IOV InfiniBand network attachment](#)

- [Using CPU Manager](#)

- [Exclude SR-IOV network topology for NUMA-aware scheduling](#)

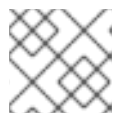## 26.8. CONFIGURING INTERFACE-LEVEL NETWORK SYSCTL SETTINGS FOR SR-IOV NETWORKS

As a cluster administrator, you can modify interface-level network sysctls using the tuning Container Network Interface (CNI) meta plugin for a pod connected to a SR-IOV network device.

26.8.1. Labeling nodes with an SR-IOV enabled NIC

If you want to enable SR-IOV on only SR-IOV capable nodes there are a couple of ways to do this:

1. Install the Node Feature Discovery (NFD) Operator. NFD detects the presence of SR-IOV enabled NICs and labels the nodes with **node.alpha.kubernetes-incubator.io/nfd-network-sriov.capable = true**.

2. Examine the **SriovNetworkNodeState** CR for each node. The **interfaces** stanza includes a list of all of the SR-IOV devices discovered by the SR-IOV Network Operator on the worker node. Label each node with **feature.node.kubernetes.io/network-sriov.capable: "true"** by using the following command:

   ```
   $ oc label node <node_name> feature.node.kubernetes.io/network-sriov.capable="true"
   ```

   > **NOTE**
   >
   > You can label the nodes with whatever name you want.

26.8.2. Setting one sysctl flag

You can set interface-level network **sysctl** settings for a pod connected to a SR-IOV network device.

In this example, **net.ipv4.conf.IFNAME.accept_redirects** is set to **1** on the created virtual interfaces.

The **sysctl-tuning-test** is a namespace used in this example.

- Use the following command to create the **sysctl-tuning-test** namespace:

```
$ oc create namespace sysctl-tuning-test
```

26.8.2.1. Setting one sysctl flag on nodes with SR-IOV network devices

The SR-IOV Network Operator adds the **SriovNetworkNodePolicy.sriovnetwork.openshift.io** custom resource definition (CRD) to OpenShift Container Platform. You can configure an SR-IOV network device by creating a **SriovNetworkNodePolicy** custom resource (CR).

> **NOTE**
>
> When applying the configuration specified in a **SriovNetworkNodePolicy** object, the SR-IOV Operator might drain and reboot the nodes.
>
> It can take several minutes for a configuration change to apply.

Follow this procedure to create a **SriovNetworkNodePolicy** custom resource (CR).

Procedure

1. Create an **SriovNetworkNodePolicy** custom resource (CR). For example, save the following YAML as the file **policyoneflag-sriov-node-network.yaml**:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: policyoneflag 1
  namespace: openshift-sriov-network-operator 2
spec:
  resourceName: policyoneflag 3
  nodeSelector: 4
    feature.node.kubernetes.io/network-sriov.capable="true"
  priority: 10 5
  numVfs: 5 6
  nicSelector: 7
    pfNames: ["ens5"] 8
  deviceType: "netdevice" 9
  isRdma: false 10
```

   1 The name for the custom resource object.

   2 The namespace where the SR-IOV Network Operator is installed.

   3 The resource name of the SR-IOV network device plugin. You can create multiple SR-IOV network node policies for a resource name.

4 The node selector specifies the nodes to configure. Only SR-IOV network devices on the selected nodes are configured. The SR-IOV Container Network Interface (CNI) plugin and

5 Optional: The priority is an integer value between **0** and **99**. A smaller value receives higher priority. For example, a priority of **10** is a higher priority than **99**. The default value is **99**.

6 The number of the virtual functions (VFs) to create for the SR-IOV physical network device. For an Intel network interface controller (NIC), the number of VFs cannot be larger than the total VFs supported by the device. For a Mellanox NIC, the number of VFs cannot be larger than **128**.

7 The NIC selector identifies the device for the Operator to configure. You do not have to specify values for all the parameters. It is recommended to identify the network device with enough precision to avoid selecting a device unintentionally. If you specify **rootDevices**, you must also specify a value for **vendor**, **deviceID**, or **pfNames**. If you specify both **pfNames** and **rootDevices** at the same time, ensure that they refer to the same device. If you specify a value for **netFilter**, then you do not need to specify any other parameter because a network ID is unique.

8 Optional: An array of one or more physical function (PF) names for the device.

9 Optional: The driver type for the virtual functions. The only allowed value is **netdevice**. For a Mellanox NIC to work in DPDK mode on bare metal nodes, set **isRdma** to **true**.

10 Optional: Configures whether to enable remote direct memory access (RDMA) mode. The default value is **false**. If the **isRdma** parameter is set to **true**, you can continue to use the RDMA-enabled VF as a normal network device. A device can be used in either mode. Set **isRdma** to **true** and additionally set **needVhostNet** to **true** to configure a Mellanox NIC for use with Fast Datapath DPDK applications.

> **NOTE**
>
> The **vfio-pci** driver type is not supported.

2. Create the **SriovNetworkNodePolicy** object:

```
$ oc create -f policyoneflag-sriov-node-network.yaml
```

After applying the configuration update, all the pods in **sriov-network-operator** namespace change to the **Running** status.

3. To verify that the SR-IOV network device is configured, enter the following command. Replace **<node_name>** with the name of a node with the SR-IOV network device that you just configured.

```
$ oc get sriovnetworknodestates -n openshift-sriov-network-operator <node_name> -o
jsonpath='{.status.syncStatus}'
```

Example output

```
Succeeded
```

26.8.2.2. Configuring sysctl on a SR-IOV network

You can set interface specific **sysctl** settings on virtual interfaces created by SR-IOV by adding the tuning configuration to the optional **metaPlugins** parameter of the **SriovNetwork** resource.

The SR-IOV Network Operator manages additional network definitions. When you specify an additional SR-IOV network to create, the SR-IOV Network Operator creates the **NetworkAttachmentDefinition** custom resource (CR) automatically.
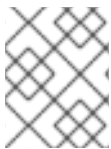
> **NOTE**
>
> Do not edit **NetworkAttachmentDefinition** custom resources that the SR-IOV Network Operator manages. Doing so might disrupt network traffic on your additional network.

To change the interface-level network **net.ipv4.conf.IFNAME.accept_redirects sysctl** settings, create an additional SR-IOV network with the Container Network Interface (CNI) tuning plugin.

Prerequisites

- Install the OpenShift Container Platform CLI (oc).

- Log in to the OpenShift Container Platform cluster as a user with cluster-admin privileges.

Procedure

1. Create the **SriovNetwork** custom resource (CR) for the additional SR-IOV network attachment and insert the **metaPlugins** configuration, as in the following example CR. Save the YAML as the file **sriov-network-interface-sysctl.yaml**.

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: onevalidflag 1
  namespace: openshift-sriov-network-operator 2
spec:
  resourceName: policyoneflag 3
  networkNamespace: sysctl-tuning-test 4
  ipam: '{ "type": "static" }' 5
  capabilities: '{ "mac": true, "ips": true }' 6
  metaPlugins : | 7
   {
     "type": "tuning",
     "capabilities":{
       "mac":true
     },
     "sysctl":{
       "net.ipv4.conf.IFNAME.accept_redirects": "1"
     }
   }
```

1 A name for the object. The SR-IOV Network Operator creates a NetworkAttachmentDefinition object with same name.

2 The namespace where the SR-IOV Network Operator is installed.

3 The value for the **spec.resourceName** parameter from the **SriovNetworkNodePolicy**

4   The target namespace for the **SriovNetwork** object. Only pods in the target namespace can attach to the additional network.

5   A configuration object for the IPAM CNI plugin as a YAML block scalar. The plugin manages IP address assignment for the attachment definition.

6   Optional: Set capabilities for the additional network. You can specify **"{ "ips": true }"** to enable IP address support or **"{ "mac": true }"** to enable MAC address support.

7   Optional: The metaPlugins parameter is used to add additional capabilities to the device. In this use case set the **type** field to **tuning**. Specify the interface-level network **sysctl** you want to set in the **sysctl** field.

2. Create the **SriovNetwork** resource:

```
$ oc create -f sriov-network-interface-sysctl.yaml
```

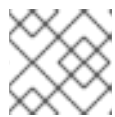**Verifying that the NetworkAttachmentDefinition CR is successfully created**

- Confirm that the SR-IOV Network Operator created the **NetworkAttachmentDefinition** CR by running the following command:

```
$ oc get network-attachment-definitions -n <namespace>
```
1

1   Replace **<namespace>** with the value for **networkNamespace** that you specified in the **SriovNetwork** object. For example, **sysctl-tuning-test**.

Example output

```
NAME                   AGE
onevalidflag           14m
```

> **NOTE**
>
> There might be a delay before the SR-IOV Network Operator creates the CR.

Verifying that the additional SR-IOV network attachment is successful

To verify that the tuning CNI is correctly configured and the additional SR-IOV network attachment is attached, do the following:

1. Create a **Pod** CR. Save the following YAML as the file **examplepod.yaml**:

```
apiVersion: v1
kind: Pod
metadata:
 name: tunepod
 namespace: sysctl-tuning-test
 annotations:
  k8s.v1.cni.cncf.io/networks: |-
    [
      {
```

```
          "name": "onevalidflag", ①
          "mac": "0a:56:0a:83:04:0c", ②
          "ips": ["10.100.100.200/24"] ③
        }
      ]
  spec:
    containers:
    - name: podexample
      image: centos
      command: ["/bin/bash", "-c", "sleep INF"]
      securityContext:
        runAsUser: 2000
        runAsGroup: 3000
        allowPrivilegeEscalation: false
        capabilities:
          drop: ["ALL"]
    securityContext:
      runAsNonRoot: true
      seccompProfile:
        type: RuntimeDefault
```

① The name of the SR-IOV network attachment definition CR.

② Optional: The MAC address for the SR-IOV device that is allocated from the resource type defined in the SR-IOV network attachment definition CR. To use this feature, you also must specify **{ "mac": true }** in the SriovNetwork object.

③ Optional: IP addresses for the SR-IOV device that are allocated from the resource type defined in the SR-IOV network attachment definition CR. Both IPv4 and IPv6 addresses are supported. To use this feature, you also must specify **{ "ips": true }** in the **SriovNetwork** object.

2. Create the **Pod** CR:

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

3. Verify that the pod is created by running the following command:

   ```
   $ oc get pod -n sysctl-tuning-test
   ```

   Example output

   ```
   NAME     READY  STATUS   RESTARTS  AGE
   tunepod  1/1    Running  0         47s
   ```

4. Log in to the pod by running the following command:

   ```
   $ oc rsh -n sysctl-tuning-test tunepod
   ```

5. Verify the values of the configured sysctl flag. Find the value **net.ipv4.conf.IFNAME.accept_redirects** by running the following command::

   ```
   $ sysctl net.ipv4.conf.net1.accept_redirects
   ```

Example output

```
net.ipv4.conf.net1.accept_redirects = 1
```

26.8.3. Configuring sysctl settings for pods associated with bonded SR-IOV interface flag

You can set interface-level network **sysctl** settings for a pod connected to a bonded SR-IOV network device.

In this example, the specific network interface-level **sysctl** settings that can be configured are set on the bonded interface.
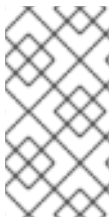
The **sysctl-tuning-test** is a namespace used in this example.

- Use the following command to create the **sysctl-tuning-test** namespace:

```
$ oc create namespace sysctl-tuning-test
```

26.8.3.1. Setting all sysctl flag on nodes with bonded SR-IOV network devices

The SR-IOV Network Operator adds the **SriovNetworkNodePolicy.sriovnetwork.openshift.io** custom resource definition (CRD) to OpenShift Container Platform. You can configure an SR-IOV network device by creating a **SriovNetworkNodePolicy** custom resource (CR).

> **NOTE**
>
> When applying the configuration specified in a SriovNetworkNodePolicy object, the SR-IOV Operator might drain the nodes, and in some cases, reboot nodes.
>
> It might take several minutes for a configuration change to apply.

Follow this procedure to create a **SriovNetworkNodePolicy** custom resource (CR).

Procedure

1. Create an **SriovNetworkNodePolicy** custom resource (CR). Save the following YAML as the file **policyallflags-sriov-node-network.yaml**. Replace **policyallflags** with the name for the configuration.
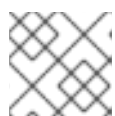
```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: policyallflags 1
  namespace: openshift-sriov-network-operator 2
spec:
  resourceName: policyallflags 3
  nodeSelector: 4
    node.alpha.kubernetes-incubator.io/nfd-network-sriov.capable = `true`
  priority: 10 5
  numVfs: 5 6
  nicSelector: 7
```

```
    pfNames: ["ens1f0"] 8
  deviceType: "netdevice" 9
  isRdma: false 10
```

1   The name for the custom resource object.

2   The namespace where the SR-IOV Network Operator is installed.

3   The resource name of the SR-IOV network device plugin. You can create multiple SR-IOV network node policies for a resource name.

4   The node selector specifies the nodes to configure. Only SR-IOV network devices on the selected nodes are configured. The SR-IOV Container Network Interface (CNI) plugin and device plugin are deployed on selected nodes only.

5   Optional: The priority is an integer value between **0** and **99**. A smaller value receives higher priority. For example, a priority of **10** is a higher priority than **99**. The default value is **99**.

6   The number of virtual functions (VFs) to create for the SR-IOV physical network device. For an Intel network interface controller (NIC), the number of VFs cannot be larger than the total VFs supported by the device. For a Mellanox NIC, the number of VFs cannot be larger than **128**.

7   The NIC selector identifies the device for the Operator to configure. You do not have to specify values for all the parameters. It is recommended to identify the network device with enough precision to avoid selecting a device unintentionally. If you specify **rootDevices**, you must also specify a value for **vendor**, **deviceID**, or **pfNames**. If you specify both **pfNames** and **rootDevices** at the same time, ensure that they refer to the same device. If you specify a value for **netFilter**, then you do not need to specify any other parameter because a network ID is unique.

8   Optional: An array of one or more physical function (PF) names for the device.

9   Optional: The driver type for the virtual functions. The only allowed value is **netdevice**. For a Mellanox NIC to work in DPDK mode on bare metal nodes, set **isRdma** to **true**.

10   Optional: Configures whether to enable remote direct memory access (RDMA) mode. The default value is **false**. If the **isRdma** parameter is set to **true**, you can continue to use the RDMA-enabled VF as a normal network device. A device can be used in either mode. Set **isRdma** to **true** and additionally set **needVhostNet** to **true** to configure a Mellanox NIC for use with Fast Datapath DPDK applications.

> **NOTE**
>
> The **vfio-pci** driver type is not supported.

2. Create the SriovNetworkNodePolicy object:

```
$ oc create -f policyallflags-sriov-node-network.yaml
```

After applying the configuration update, all the pods in sriov-network-operator namespace change to the **Running** status.

3. To verify that the SR-IOV network device is configured, enter the following command. Replace **<node_name>** with the name of a node with the SR-IOV network device that you just configured.

```
$ oc get sriovnetworknodestates -n openshift-sriov-network-operator <node_name> -o
jsonpath='{.status.syncStatus}'
```

Example output

```
Succeeded
```

26.8.3.2. Configuring sysctl on a bonded SR-IOV network

You can set interface specific **sysctl** settings on a bonded interface created from two SR-IOV interfaces. Do this by adding the tuning configuration to the optional **Plugins** parameter of the bond network attachment definition.

> **NOTE**
>
> Do not edit **NetworkAttachmentDefinition** custom resources that the SR-IOV Network Operator manages. Doing so might disrupt network traffic on your additional network.

To change specific interface-level network **sysctl** settings create the **SriovNetwork** custom resource (CR) with the Container Network Interface (CNI) tuning plugin by using the following procedure.

Prerequisites

- Install the OpenShift Container Platform CLI (oc).

- Log in to the OpenShift Container Platform cluster as a user with cluster-admin privileges.

Procedure

1. Create the **SriovNetwork** custom resource (CR) for the bonded interface as in the following example CR. Save the YAML as the file **sriov-network-attachment.yaml**.

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: allvalidflags  1
  namespace: openshift-sriov-network-operator  2
spec:
  resourceName: policyallflags  3
  networkNamespace: sysctl-tuning-test  4
  capabilities: '{ "mac": true, "ips": true }'  5
```

1 A name for the object. The SR-IOV Network Operator creates a NetworkAttachmentDefinition object with same name.

2 The namespace where the SR-IOV Network Operator is installed.

3 The value for the **spec.resourceName** parameter from the **SriovNetworkNodePolicy** object that defines the SR-IOV hardware for this additional network.

4 The target namespace for the **SriovNetwork** object. Only pods in the target namespace can attach to the additional network.

5 Optional: The capabilities to configure for this additional network. You can specify **"{ "ips": true }"** to enable IP address support or **"{ "mac": true }"** to enable MAC address support.

2. Create the **SriovNetwork** resource:

```
$ oc create -f sriov-network-attachment.yaml
```

3. Create a bond network attachment definition as in the following example CR. Save the YAML as the file **sriov-bond-network-interface.yaml**.

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: bond-sysctl-network
  namespace: sysctl-tuning-test
spec:
  config: '{
  "cniVersion":"0.4.0",
  "name":"bound-net",
  "plugins":[
   {
     "type":"bond", 1
     "mode": "active-backup", 2
     "failOverMac": 1, 3
     "linksInContainer": true, 4
     "miimon": "100",
     "links": [ 5
       {"name": "net1"},
       {"name": "net2"}
     ],
     "ipam":{ 6
       "type":"static"
     }
   },
   {
     "type":"tuning", 7
     "capabilities":{
       "mac":true
     },
     "sysctl":{
       "net.ipv4.conf.IFNAME.accept_redirects": "0",
       "net.ipv4.conf.IFNAME.accept_source_route": "0",
       "net.ipv4.conf.IFNAME.disable_policy": "1",
       "net.ipv4.conf.IFNAME.secure_redirects": "0",
       "net.ipv4.conf.IFNAME.send_redirects": "0",
       "net.ipv6.conf.IFNAME.accept_redirects": "0",
       "net.ipv6.conf.IFNAME.accept_source_route": "1",
       "net.ipv6.neigh.IFNAME.base_reachable_time_ms": "20000",
       "net.ipv6.neigh.IFNAME.retrans_time_ms": "2000"
     }
```

```
      }
    ]
  }'
```

1   The type is **bond**.

2   The **mode** attribute specifies the bonding mode. The bonding modes supported are:

- **balance-rr** - 0

- **active-backup** - 1

- **balance-xor** - 2
  For **balance-rr** or **balance-xor** modes, you must set the **trust** mode to **on** for the SR-IOV virtual function.

3   The **failover** attribute is mandatory for active-backup mode.

4   The **linksInContainer=true** flag informs the Bond CNI that the required interfaces are to be found inside the container. By default, Bond CNI looks for these interfaces on the host which does not work for integration with SRIOV and Multus.

5   The **links** section defines which interfaces will be used to create the bond. By default, Multus names the attached interfaces as: "net", plus a consecutive number, starting with one.

6   A configuration object for the IPAM CNI plugin as a YAML block scalar. The plugin manages IP address assignment for the attachment definition. In this pod example IP addresses are configured manually, so in this case,**ipam** is set to static.

7   Add additional capabilities to the device. For example, set the **type** field to **tuning**. Specify the interface-level network **sysctl** you want to set in the sysctl field. This example sets all interface-level network **sysctl** settings that can be set.

4. Create the bond network attachment resource:

```
$ oc create -f sriov-bond-network-interface.yaml
```

Verifying that the NetworkAttachmentDefinition CR is successfully created

- Confirm that the SR-IOV Network Operator created the **NetworkAttachmentDefinition** CR by running the following command:

```
$ oc get network-attachment-definitions -n <namespace>
```
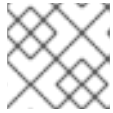**1**

1   Replace **<namespace>** with the networkNamespace that you specified when configuring the network attachment, for example, **sysctl-tuning-test**.

Example output

```
NAME                 AGE
bond-sysctl-network       22m
allvalidflags          47m
```

> **NOTE**
>
> There might be a delay before the SR-IOV Network Operator creates the CR.

Verifying that the additional SR-IOV network resource is successful

To verify that the tuning CNI is correctly configured and the additional SR-IOV network attachment is attached, do the following:

1. Create a **Pod** CR. For example, save the following YAML as the file **examplepod.yaml**:

```
apiVersion: v1
kind: Pod
metadata:
  name: tunepod
  namespace: sysctl-tuning-test
  annotations:
    k8s.v1.cni.cncf.io/networks: |-
      [
        {"name": "allvalidflags"}, ❶
        {"name": "allvalidflags"},
        {
          "name": "bond-sysctl-network",
          "interface": "bond0",
          "mac": "0a:56:0a:83:04:0c", ❷
          "ips": ["10.100.100.200/24"] ❸
        }
      ]
spec:
  containers:
  - name: podexample
    image: centos
    command: ["/bin/bash", "-c", "sleep INF"]
    securityContext:
      runAsUser: 2000
      runAsGroup: 3000
      allowPrivilegeEscalation: false
      capabilities:
        drop: ["ALL"]
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
```

❶ The name of the SR-IOV network attachment definition CR.

❷ Optional: The MAC address for the SR-IOV device that is allocated from the resource type defined in the SR-IOV network attachment definition CR. To use this feature, you also must specify **{ "mac": true }** in the SriovNetwork object.

❸ Optional: IP addresses for the SR-IOV device that are allocated from the resource type defined in the SR-IOV network attachment definition CR. Both IPv4 and IPv6 addresses are supported. To use this feature, you also must specify **{ "ips": true }** in the **SriovNetwork** object.

2. Apply the YAML:

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

3. Verify that the pod is created by running the following command:

   ```
   $ oc get pod -n sysctl-tuning-test
   ```

   Example output

   ```
   NAME      READY  STATUS   RESTARTS  AGE
   tunepod   1/1    Running  0         47s
   ```

4. Log in to the pod by running the following command:

   ```
   $ oc rsh -n sysctl-tuning-test tunepod
   ```

5. Verify the values of the configured **sysctl** flag. Find the value
   net.ipv6.neigh.IFNAME.base_reachable_time_ms by running the following command::

   ```
   $ sysctl net.ipv6.neigh.bond0.base_reachable_time_ms
   ```

   Example output

   ```
   net.ipv6.neigh.bond0.base_reachable_time_ms = 20000
   ```

26.9. USING HIGH PERFORMANCE MULTICAST

You can use multicast on your Single Root I/O Virtualization (SR-IOV) hardware network.

26.9.1. High performance multicast

The OpenShift SDN network plugin supports multicast between pods on the default network. This is best used for low-bandwidth coordination or service discovery, and not high-bandwidth applications. For applications such as streaming media, like Internet Protocol television (IPTV) and multipoint videoconferencing, you can utilize Single Root I/O Virtualization (SR-IOV) hardware to provide near-native performance.

When using additional SR-IOV interfaces for multicast:

- Multicast packages must be sent or received by a pod through the additional SR-IOV interface.

- The physical network which connects the SR-IOV interfaces decides the multicast routing and topology, which is not controlled by OpenShift Container Platform.

26.9.2. Configuring an SR-IOV interface for multicast

The follow procedure creates an example SR-IOV interface for multicast.

Prerequisites

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

- You must log in to the cluster with a user that has the **cluster-admin** role.

Procedure

1. Create a **SriovNetworkNodePolicy** object:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: policy-example
  namespace: openshift-sriov-network-operator
spec:
  resourceName: example
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  numVfs: 4
  nicSelector:
    vendor: "8086"
    pfNames: ['ens803f0']
    rootDevices: ['0000:86:00.0']
```

2. Create a **SriovNetwork** object:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: net-example
  namespace: openshift-sriov-network-operator
spec:
  networkNamespace: default
  ipam: | ❶
    {
      "type": "host-local", ❷
      "subnet": "10.56.217.0/24",
      "rangeStart": "10.56.217.171",
      "rangeEnd": "10.56.217.181",
      "routes": [
        {"dst": "224.0.0.0/5"},
        {"dst": "232.0.0.0/5"}
      ],
      "gateway": "10.56.217.1"
    }
  resourceName: example
```

❶ ❷ If you choose to configure DHCP as IPAM, ensure that you provision the following default routes through your DHCP server: **224.0.0.0/5** and **232.0.0.0/5**. This is to override the static multicast route set by the default network provider.

3. Create a pod with multicast application:

```
apiVersion: v1
kind: Pod
metadata:
  name: testpmd
```

```
   namespace: default
   annotations:
     k8s.v1.cni.cncf.io/networks: nic1
 spec:
  containers:
  - name: example
    image: rhel7:latest
    securityContext:
      capabilities:
        add: ["NET_ADMIN"] ❶
    command: [ "sleep", "infinity"]
```

❶ The **NET_ADMIN** capability is required only if your application needs to assign the multicast IP address to the SR-IOV interface. Otherwise, it can be omitted.

26.10. USING DPDK AND RDMA

The containerized Data Plane Development Kit (DPDK) application is supported on OpenShift Container Platform. You can use Single Root I/O Virtualization (SR-IOV) network hardware with the Data Plane Development Kit (DPDK) and with remote direct memory access (RDMA).

For information on supported devices, refer to Supported devices.

26.10.1. Using a virtual function in DPDK mode with an Intel NIC

Prerequisites

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

- Install the SR-IOV Network Operator.

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

Procedure

1. Create the following **SriovNetworkNodePolicy** object, and then save the YAML in the **intel-dpdk-node-policy.yaml** file.

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: intel-dpdk-node-policy
  namespace: openshift-sriov-network-operator
spec:
  resourceName: intelnics
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  priority: <priority>
  numVfs: <num>
  nicSelector:
    vendor: "8086"
    deviceID: "158b"
```

```
    pfNames: ["<pf_name>", ...]
    rootDevices: ["<pci_bus_id>", "..."]
  deviceType: vfio-pci 1
```

1 Specify the driver type for the virtual functions to **vfio-pci**.

> **NOTE**
>
> See the **Configuring SR-IOV network devices** section for a detailed explanation on each option in **SriovNetworkNodePolicy**.
>
> When applying the configuration specified in a **SriovNetworkNodePolicy** object, the SR-IOV Operator may drain the nodes, and in some cases, reboot nodes. It may take several minutes for a configuration change to apply. Ensure that there are enough available nodes in your cluster to handle the evicted workload beforehand.
>
> After the configuration update is applied, all the pods in **openshift-sriov-network-operator** namespace will change to a **Running** status.

2. Create the **SriovNetworkNodePolicy** object by running the following command:

   ```
   $ oc create -f intel-dpdk-node-policy.yaml
   ```

3. Create the following **SriovNetwork** object, and then save the YAML in the **intel-dpdk-network.yaml** file.

   ```
   apiVersion: sriovnetwork.openshift.io/v1
   kind: SriovNetwork
   metadata:
     name: intel-dpdk-network
     namespace: openshift-sriov-network-operator
   spec:
     networkNamespace: <target_namespace>
     ipam: |-
   # ... 1
     vlan: <vlan>
     resourceName: intelnics
   ```

   1 Specify a configuration object for the ipam CNI plugin as a YAML block scalar. The plugin manages IP address assignment for the attachment definition.

   > **NOTE**
   >
   > See the "Configuring SR-IOV additional network" section for a detailed explanation on each option in **SriovNetwork**.

   An optional library, app-netutil, provides several API methods for gathering network information about a container's parent pod.

4. Create the **SriovNetwork** object by running the following command:

```
$ oc create -f intel-dpdk-network.yaml
```

5. Create the following **Pod** spec, and then save the YAML in the **intel-dpdk-pod.yaml** file.

```
apiVersion: v1
kind: Pod
metadata:
  name: dpdk-app
  namespace: <target_namespace>          1
  annotations:
    k8s.v1.cni.cncf.io/networks: intel-dpdk-network
spec:
  containers:
  - name: testpmd
    image: <DPDK_image>                   2
    securityContext:
      runAsUser: 0
      capabilities:
        add: ["IPC_LOCK","SYS_RESOURCE","NET_RAW"]   3
    volumeMounts:
    - mountPath: /mnt/huge                 4
      name: hugepage
    resources:
      limits:
        openshift.io/intelnics: "1"        5
        memory: "1Gi"
        cpu: "4"                           6
        hugepages-1Gi: "4Gi"               7
      requests:
        openshift.io/intelnics: "1"
        memory: "1Gi"
        cpu: "4"
        hugepages-1Gi: "4Gi"
    command: ["sleep", "infinity"]
  volumes:
  - name: hugepage
    emptyDir:
      medium: HugePages
```

1 Specify the same **target_namespace** where the **SriovNetwork** object **intel-dpdk-network** is created. If you would like to create the pod in a different namespace, change **target_namespace** in both the **Pod** spec and the **SriovNetwork** object.

2 Specify the DPDK image which includes your application and the DPDK library used by application.

3 Specify additional capabilities required by the application inside the container for hugepage allocation, system resource allocation, and network interface access.

4 Mount a hugepage volume to the DPDK pod under **/mnt/huge**. The hugepage volume is backed by the emptyDir volume type with the medium being **Hugepages**.

5 Optional: Specify the number of DPDK devices allocated to DPDK pod. This resource request and limit, if not explicitly specified, will be automatically added by the SR-IOV network resource injector. The SR-IOV network resource injector is an admission controller

component managed by the SR-IOV Operator. It is enabled by default and can be disabled by setting **enableInjector** option to **false** in the default **SriovOperatorConfig** CR.

6 Specify the number of CPUs. The DPDK pod usually requires exclusive CPUs to be allocated from the kubelet. This is achieved by setting CPU Manager policy to **static** and creating a pod with **Guaranteed** QoS.

7 Specify hugepage size **hugepages-1Gi** or **hugepages-2Mi** and the quantity of hugepages that will be allocated to the DPDK pod. Configure **2Mi** and **1Gi** hugepages separately. Configuring **1Gi** hugepage requires adding kernel arguments to Nodes. For example, adding kernel arguments **default_hugepagesz=1GB**, **hugepagesz=1G** and **hugepages=16** will result in **16*1Gi** hugepages be allocated during system boot.

6. Create the DPDK pod by running the following command:

```
$ oc create -f intel-dpdk-pod.yaml
```

26.10.2. Using a virtual function in DPDK mode with a Mellanox NIC

You can create a network node policy and create a Data Plane Development Kit (DPDK) pod using a virtual function in DPDK mode with a Mellanox NIC.

Prerequisites

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

- You have installed the Single Root I/O Virtualization (SR-IOV) Network Operator.

- You have logged in as a user with **cluster-admin** privileges.

Procedure

1. Save the following **SriovNetworkNodePolicy** YAML configuration to an **mlx-dpdk-node-policy.yaml** file:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: mlx-dpdk-node-policy
  namespace: openshift-sriov-network-operator
spec:
  resourceName: mlxnics
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  priority: <priority>
  numVfs: <num>
  nicSelector:
    vendor: "15b3"
    deviceID: "1015" 1
    pfNames: ["<pf_name>", ...]
    rootDevices: ["<pci_bus_id>", "..."]
  deviceType: netdevice 2
  isRdma: true 3
```

1 Specify the device hex code of the SR-IOV network device.

2 Specify the driver type for the virtual functions to **netdevice**. A Mellanox SR-IOV Virtual Function (VF) can work in DPDK mode without using the **vfio-pci** device type. The VF device appears as a kernel network interface inside a container.

3 Enable Remote Direct Memory Access (RDMA) mode. This is required for Mellanox cards to work in DPDK mode.

> **NOTE**
>
> See *Configuring an SR-IOV network device* for a detailed explanation of each option in the **SriovNetworkNodePolicy** object.
>
> When applying the configuration specified in an **SriovNetworkNodePolicy** object, the SR-IOV Operator might drain the nodes, and in some cases, reboot nodes. It might take several minutes for a configuration change to apply. Ensure that there are enough available nodes in your cluster to handle the evicted workload beforehand.
>
> After the configuration update is applied, all the pods in the **openshift-sriov-network-operator** namespace will change to a **Running** status.

2. Create the **SriovNetworkNodePolicy** object by running the following command:

   ```
   $ oc create -f mlx-dpdk-node-policy.yaml
   ```

3. Save the following **SriovNetwork** YAML configuration to an **mlx-dpdk-network.yaml** file:

   ```
   apiVersion: sriovnetwork.openshift.io/v1
   kind: SriovNetwork
   metadata:
     name: mlx-dpdk-network
     namespace: openshift-sriov-network-operator
   spec:
     networkNamespace: <target_namespace>
     ipam: |- 1
   ...
     vlan: <vlan>
     resourceName: mlxnics
   ```

1 Specify a configuration object for the IP Address Management (IPAM) Container Network Interface (CNI) plugin as a YAML block scalar. The plugin manages IP address assignment for the attachment definition.

> **NOTE**
>
> See *Configuring an SR-IOV network device* for a detailed explanation on each option in the **SriovNetwork** object.

The **app-netutil** option library provides several API methods for gathering network information about the parent pod of a container.

4. Create the **SriovNetwork** object by running the following command:

```
$ oc create -f mlx-dpdk-network.yaml
```

5. Save the following **Pod** YAML configuration to an **mlx-dpdk-pod.yaml** file:

```
apiVersion: v1
kind: Pod
metadata:
  name: dpdk-app
  namespace: <target_namespace> 1
  annotations:
    k8s.v1.cni.cncf.io/networks: mlx-dpdk-network
spec:
  containers:
  - name: testpmd
    image: <DPDK_image> 2
    securityContext:
      runAsUser: 0
      capabilities:
        add: ["IPC_LOCK","SYS_RESOURCE","NET_RAW"] 3
    volumeMounts:
    - mountPath: /mnt/huge 4
      name: hugepage
    resources:
      limits:
        openshift.io/mlxnics: "1" 5
        memory: "1Gi"
        cpu: "4" 6
        hugepages-1Gi: "4Gi" 7
      requests:
        openshift.io/mlxnics: "1"
        memory: "1Gi"
        cpu: "4"
        hugepages-1Gi: "4Gi"
    command: ["sleep", "infinity"]
  volumes:
  - name: hugepage
    emptyDir:
      medium: HugePages
```

1 Specify the same **target_namespace** where **SriovNetwork** object **mlx-dpdk-network** is created. To create the pod in a different namespace, change **target_namespace** in both the **Pod** spec and **SriovNetwork** object.

2 Specify the DPDK image which includes your application and the DPDK library used by the application.

3 Specify additional capabilities required by the application inside the container for hugepage allocation, system resource allocation, and network interface access.

4 Mount the hugepage volume to the DPDK pod under **/mnt/huge**. The hugepage volume is backed by the **emptyDir** volume type with the medium being **Hugepages**.

5 Optional: Specify the number of DPDK devices allocated for the DPDK pod. If not explicitly specified, this resource request and limit is automatically added by the SR-IOV network

6 Specify the number of CPUs. The DPDK pod usually requires that exclusive CPUs be allocated from the kubelet. To do this, set the CPU Manager policy to **static** and create a pod with **Guaranteed** Quality of Service (QoS).

7 Specify hugepage size **hugepages-1Gi** or **hugepages-2Mi** and the quantity of hugepages that will be allocated to the DPDK pod. Configure **2Mi** and **1Gi** hugepages separately. Configuring **1Gi** hugepages requires adding kernel arguments to Nodes.

6. Create the DPDK pod by running the following command:

```
$ oc create -f mlx-dpdk-pod.yaml
```

26.10.3. Overview of achieving a specific DPDK line rate

To achieve a specific Data Plane Development Kit (DPDK) line rate, deploy a Node Tuning Operator and configure Single Root I/O Virtualization (SR-IOV). You must also tune the DPDK settings for the following resources:

- Isolated CPUs

- Hugepages

- The topology scheduler

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

DPDK test environment

The following diagram shows the components of a traffic-testing environment:
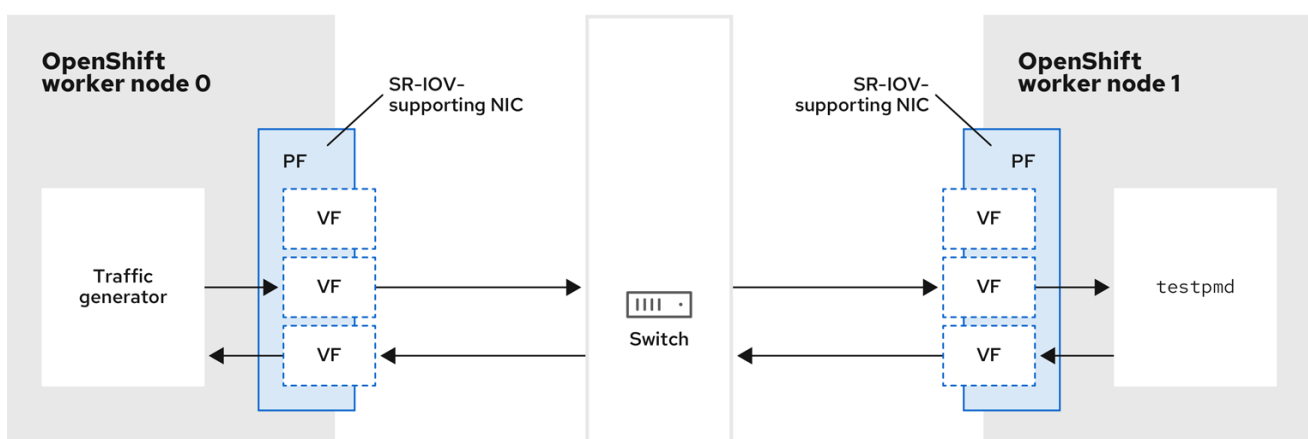
- **Traffic generator**: An application that can generate high-volume packet traffic.

- **SR-IOV-supporting NIC**: A network interface card compatible with SR-IOV. The card runs a number of virtual functions on a physical interface.

- **Physical Function (PF)**: A PCI Express (PCIe) function of a network adapter that supports the SR-IOV interface.

- **Virtual Function (VF)**: A lightweight PCIe function on a network adapter that supports SR-IOV. The VF is associated with the PCIe PF on the network adapter. The VF represents a virtualized instance of the network adapter.

- **Switch**: A network switch. Nodes can also be connected back-to-back.

- **testpmd**: An example application included with DPDK. The **testpmd** application can be used to test the DPDK in a packet-forwarding mode. The **testpmd** application is also an example of how to build a fully-fledged application using the DPDK Software Development Kit (SDK).

- **worker 0** and **worker 1**: OpenShift Container Platform nodes.

26.10.4. Using SR-IOV and the Node Tuning Operator to achieve a DPDK line rate

You can use the Node Tuning Operator to configure isolated CPUs, hugepages, and a topology scheduler. You can then use the Node Tuning Operator with Single Root I/O Virtualization (SR-IOV) to achieve a specific Data Plane Development Kit (DPDK) line rate.

Prerequisites

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

- You have installed the SR-IOV Network Operator.

- You have logged in as a user with **cluster-admin** privileges.

- You have deployed a standalone Node Tuning Operator.

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

Procedure

1. Create a **PerformanceProfile** object based on the following example:

   ```
   apiVersion: performance.openshift.io/v2
   kind: PerformanceProfile
   metadata:
     name: performance
   spec:
     globallyDisableIrqLoadBalancing: true
     cpu:
       isolated: 21-51,73-103 1
   ```

```
      reserved: 0-20,52-72 2
    hugepages:
      defaultHugepagesSize: 1G 3
      pages:
        - count: 32
          size: 1G
    net:
      userLevelNetworking: true
    numa:
      topologyPolicy: "single-numa-node"
    nodeSelector:
      node-role.kubernetes.io/worker-cnf: ""
```

1 If hyperthreading is enabled on the system, allocate the relevant symbolic links to the **isolated** and **reserved** CPU groups. If the system contains multiple non-uniform memory access nodes (NUMAs), allocate CPUs from both NUMAs to both groups. You can also use the Performance Profile Creator for this task. For more information, see *Creating a performance profile*.

2 You can also specify a list of devices that will have their queues set to the reserved CPU count. For more information, see *Reducing NIC queues using the Node Tuning Operator* .

3 Allocate the number and size of hugepages needed. You can specify the NUMA configuration for the hugepages. By default, the system allocates an even number to every NUMA node on the system. If needed, you can request the use of a realtime kernel for the nodes. See *Provisioning a worker with real-time capabilities* for more information.

2. Save the **yaml** file as **mlx-dpdk-perfprofile-policy.yaml**.

3. Apply the performance profile using the following command:

```
$ oc create -f mlx-dpdk-perfprofile-policy.yaml
```

26.10.4.1. Example SR-IOV Network Operator for virtual functions

You can use the Single Root I/O Virtualization (SR-IOV) Network Operator to allocate and configure Virtual Functions (VFs) from SR-IOV-supporting Physical Function NICs on the nodes.

For more information on deploying the Operator, see *Installing the SR-IOV Network Operator* . For more information on configuring an SR-IOV network device, see *Configuring an SR-IOV network device* .

There are some differences between running Data Plane Development Kit (DPDK) workloads on Intel VFs and Mellanox VFs. This section provides object configuration examples for both VF types. The following is an example of an **sriovNetworkNodePolicy** object used to run DPDK applications on Intel NICs:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: dpdk-nic-1
  namespace: openshift-sriov-network-operator
spec:
  deviceType: vfio-pci 1
  needVhostNet: true 2
```

```
  nicSelector:
    pfNames: ["ens3f0"]
  nodeSelector:
    node-role.kubernetes.io/worker-cnf: ""
  numVfs: 10
  priority: 99
  resourceName: dpdk_nic_1
---
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: dpdk-nic-1
  namespace: openshift-sriov-network-operator
spec:
  deviceType: vfio-pci
  needVhostNet: true
  nicSelector:
    pfNames: ["ens3f1"]
  nodeSelector:
  node-role.kubernetes.io/worker-cnf: ""
  numVfs: 10
  priority: 99
  resourceName: dpdk_nic_2
```

[1] For Intel NICs, **deviceType** must be **vfio-pci**.

[2] If kernel communication with DPDK workloads is required, add **needVhostNet: true**. This mounts
the /**dev**/**net**/**tun** and /**dev**/**vhost-net** devices into the container so the application can create a tap
device and connect the tap device to the DPDK workload.

The following is an example of an **sriovNetworkNodePolicy** object for Mellanox NICs:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: dpdk-nic-1
  namespace: openshift-sriov-network-operator
spec:
  deviceType: netdevice    1
  isRdma: true    2
  nicSelector:
    rootDevices:
      - "0000:5e:00.1"
  nodeSelector:
    node-role.kubernetes.io/worker-cnf: ""
  numVfs: 5
  priority: 99
  resourceName: dpdk_nic_1
---
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: dpdk-nic-2
  namespace: openshift-sriov-network-operator
spec:
```

```
        deviceType: netdevice
        isRdma: true
        nicSelector:
          rootDevices:
            - "0000:5e:00.0"
        nodeSelector:
          node-role.kubernetes.io/worker-cnf: ""
        numVfs: 5
        priority: 99
        resourceName: dpdk_nic_2
```

1   For Mellanox devices the **deviceType** must be **netdevice**.

2   For Mellanox devices **isRdma** must be **true**. Mellanox cards are connected to DPDK applications using Flow Bifurcation. This mechanism splits traffic between Linux user space and kernel space, and can enhance line rate processing capability.

26.10.4.2. Example SR-IOV network operator

The following is an example definition of an **sriovNetwork** object. In this case, Intel and Mellanox configurations are identical:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: dpdk-network-1
  namespace: openshift-sriov-network-operator
spec:
  ipam: '{"type": "host-local","ranges": [[{"subnet": "10.0.1.0/24"}]],"dataDir":
   "/run/my-orchestrator/container-ipam-state-1"}'   1
  networkNamespace: dpdk-test   2
  spoofChk: "off"
  trust: "on"
  resourceName: dpdk_nic_1   3
---
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: dpdk-network-2
  namespace: openshift-sriov-network-operator
spec:
  ipam: '{"type": "host-local","ranges": [[{"subnet": "10.0.2.0/24"}]],"dataDir":
   "/run/my-orchestrator/container-ipam-state-1"}'
  networkNamespace: dpdk-test
  spoofChk: "off"
  trust: "on"
  resourceName: dpdk_nic_2
```

1   You can use a different IP Address Management (IPAM) implementation, such as Whereabouts. For more information, see *Dynamic IP address assignment configuration with Whereabouts* .

2   You must request the **networkNamespace** where the network attachment definition will be created. You must create the **sriovNetwork** CR under the **openshift-sriov-network-operator** namespace.

3 The **resourceName** value must match that of the **resourceName** created under the **sriovNetworkNodePolicy**.

26.10.4.3. Example DPDK base workload

The following is an example of a Data Plane Development Kit (DPDK) container:

```
apiVersion: v1
kind: Namespace
metadata:
  name: dpdk-test
---
apiVersion: v1
kind: Pod
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: '[ 1
    {
      "name": "dpdk-network-1",
      "namespace": "dpdk-test"
    },
    {
      "name": "dpdk-network-2",
      "namespace": "dpdk-test"
    }
   ]'
    irq-load-balancing.crio.io: "disable" 2
    cpu-load-balancing.crio.io: "disable"
    cpu-quota.crio.io: "disable"
  labels:
    app: dpdk
  name: testpmd
  namespace: dpdk-test
spec:
  runtimeClassName: performance-performance 3
  containers:
  - command:
      - /bin/bash
      - -c
      - sleep INF
    image: registry.redhat.io/openshift4/dpdk-base-rhel8
    imagePullPolicy: Always
    name: dpdk
    resources: 4
     limits:
       cpu: "16"
       hugepages-1Gi: 8Gi
       memory: 2Gi
     requests:
       cpu: "16"
       hugepages-1Gi: 8Gi
       memory: 2Gi
    securityContext:
     capabilities:
```

```
      add:
        - IPC_LOCK
        - SYS_RESOURCE
        - NET_RAW
        - NET_ADMIN
      runAsUser: 0
    volumeMounts:
      - mountPath: /mnt/huge
        name: hugepages
  terminationGracePeriodSeconds: 5
  volumes:
    - emptyDir:
        medium: HugePages
      name: hugepages
```

1   Request the SR-IOV networks you need. Resources for the devices will be injected automatically.

2   Disable the CPU and IRQ load balancing base. See *Disabling interrupt processing for individual pods* for more information.

3   Set the **runtimeClass** to **performance-performance**. Do not set the **runtimeClass** to **HostNetwork** or **privileged**.

4   Request an equal number of resources for requests and limits to start the pod with **Guaranteed** Quality of Service (QoS).

> **NOTE**
>
> Do not start the pod with **SLEEP** and then exec into the pod to start the testpmd or the DPDK workload. This can add additional interrupts as the **exec** process is not pinned to any CPU.

26.10.4.4. Example testpmd script

The following is an example script for running **testpmd**:

```bash
#!/bin/bash
set -ex
export CPU=$(cat /sys/fs/cgroup/cpuset/cpuset.cpus)
echo ${CPU}

dpdk-testpmd -l ${CPU} -a ${PCIDEVICE_OPENSHIFT_IO_DPDK_NIC_1} -a
${PCIDEVICE_OPENSHIFT_IO_DPDK_NIC_2} -n 4 -- -i --nb-cores=15 --rxd=4096 --txd=4096 --
rxq=7 --txq=7 --forward-mode=mac --eth-peer=0,50:00:00:00:00:01 --eth-peer=1,50:00:00:00:00:02
```

This example uses two different **sriovNetwork** CRs. The environment variable contains the Virtual Function (VF) PCI address that was allocated for the pod. If you use the same network in the pod definition, you must split the **pciAddress**. It is important to configure the correct MAC addresses of the traffic generator. This example uses custom MAC addresses.

26.10.5. Using a virtual function in RDMA mode with a Mellanox NIC

IMPORTANT

RDMA over Converged Ethernet (RoCE) is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

RDMA over Converged Ethernet (RoCE) is the only supported mode when using RDMA on OpenShift Container Platform.

Prerequisites

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

- Install the SR-IOV Network Operator.

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

Procedure

1. Create the following **SriovNetworkNodePolicy** object, and then save the YAML in the **mlx-rdma-node-policy.yaml** file.

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: mlx-rdma-node-policy
  namespace: openshift-sriov-network-operator
spec:
  resourceName: mlxnics
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  priority: <priority>
  numVfs: <num>
  nicSelector:
    vendor: "15b3"
    deviceID: "1015"  1
    pfNames: ["<pf_name>", ...]
    rootDevices: ["<pci_bus_id>", "..."]
  deviceType: netdevice  2
  isRdma: true  3
```

1	Specify the device hex code of the SR-IOV network device.
2	Specify the driver type for the virtual functions to **netdevice**.
3	Enable RDMA mode.

> **NOTE**
>
> See the **Configuring SR-IOV network devices** section for a detailed explanation on each option in **SriovNetworkNodePolicy**.
>
> When applying the configuration specified in a **SriovNetworkNodePolicy** object, the SR-IOV Operator may drain the nodes, and in some cases, reboot nodes. It may take several minutes for a configuration change to apply. Ensure that there are enough available nodes in your cluster to handle the evicted workload beforehand.
>
> After the configuration update is applied, all the pods in the **openshift-sriov-network-operator** namespace will change to a **Running** status.

2. Create the **SriovNetworkNodePolicy** object by running the following command:

   ```
   $ oc create -f mlx-rdma-node-policy.yaml
   ```

3. Create the following **SriovNetwork** object, and then save the YAML in the **mlx-rdma-network.yaml** file.

   ```
   apiVersion: sriovnetwork.openshift.io/v1
   kind: SriovNetwork
   metadata:
     name: mlx-rdma-network
     namespace: openshift-sriov-network-operator
   spec:
     networkNamespace: <target_namespace>
     ipam: |-   1
   # ...
     vlan: <vlan>
     resourceName: mlxnics
   ```

   1 Specify a configuration object for the ipam CNI plugin as a YAML block scalar. The plugin manages IP address assignment for the attachment definition.

> **NOTE**
>
> See the "Configuring SR-IOV additional network" section for a detailed explanation on each option in **SriovNetwork**.

An optional library, app-netutil, provides several API methods for gathering network information about a container's parent pod.

4. Create the **SriovNetworkNodePolicy** object by running the following command:

   ```
   $ oc create -f mlx-rdma-network.yaml
   ```

5. Create the following **Pod** spec, and then save the YAML in the **mlx-rdma-pod.yaml** file.

   ```
   apiVersion: v1
   kind: Pod
   metadata:
   ```

```
    name: rdma-app
    namespace: <target_namespace>  1
    annotations:
      k8s.v1.cni.cncf.io/networks: mlx-rdma-network
  spec:
    containers:
    - name: testpmd
      image: <RDMA_image>  2
      securityContext:
        runAsUser: 0
        capabilities:
          add: ["IPC_LOCK","SYS_RESOURCE","NET_RAW"]  3
      volumeMounts:
      - mountPath: /mnt/huge  4
        name: hugepage
      resources:
        limits:
          memory: "1Gi"
          cpu: "4"  5
          hugepages-1Gi: "4Gi"  6
        requests:
          memory: "1Gi"
          cpu: "4"
          hugepages-1Gi: "4Gi"
      command: ["sleep", "infinity"]
    volumes:
    - name: hugepage
      emptyDir:
        medium: HugePages
```

1. Specify the same **target_namespace** where **SriovNetwork** object **mlx-rdma-network** is created. If you would like to create the pod in a different namespace, change **target_namespace** in both **Pod** spec and **SriovNetwork** object.

2. Specify the RDMA image which includes your application and RDMA library used by application.

3. Specify additional capabilities required by the application inside the container for hugepage allocation, system resource allocation, and network interface access.

4. Mount the hugepage volume to RDMA pod under **/mnt/huge**. The hugepage volume is backed by the emptyDir volume type with the medium being **Hugepages**.

5. Specify number of CPUs. The RDMA pod usually requires exclusive CPUs be allocated from the kubelet. This is achieved by setting CPU Manager policy to **static** and create pod with **Guaranteed** QoS.

6. Specify hugepage size **hugepages-1Gi** or **hugepages-2Mi** and the quantity of hugepages that will be allocated to the RDMA pod. Configure **2Mi** and **1Gi** hugepages separately. Configuring **1Gi** hugepage requires adding kernel arguments to Nodes.

6. Create the RDMA pod by running the following command:

```
$ oc create -f mlx-rdma-pod.yaml
```

26.10.6. A test pod template for clusters that use OVS-DPDK on OpenStack

The following **testpmd** pod demonstrates container creation with huge pages, reserved CPUs, and the SR-IOV port.

An example testpmd pod

```
apiVersion: v1
kind: Pod
metadata:
  name: testpmd-dpdk
  namespace: mynamespace
  annotations:
    cpu-load-balancing.crio.io: "disable"
    cpu-quota.crio.io: "disable"
# ...
spec:
  containers:
  - name: testpmd
    command: ["sleep", "99999"]
    image: registry.redhat.io/openshift4/dpdk-base-rhel8:v4.9
    securityContext:
      capabilities:
        add: ["IPC_LOCK","SYS_ADMIN"]
      privileged: true
      runAsUser: 0
    resources:
      requests:
        memory: 1000Mi
        hugepages-1Gi: 1Gi
        cpu: '2'
        openshift.io/dpdk1: 1          ❶
      limits:
        hugepages-1Gi: 1Gi
        cpu: '2'
        memory: 1000Mi
        openshift.io/dpdk1: 1
    volumeMounts:
      - mountPath: /mnt/huge
        name: hugepage
        readOnly: False
  runtimeClassName: performance-cnf-performanceprofile    ❷
  volumes:
  - name: hugepage
    emptyDir:
      medium: HugePages
```

❶ The name **dpdk1** in this example is a user-created **SriovNetworkNodePolicy** resource. You can substitute this name for that of a resource that you create.

❷ If your performance profile is not named **cnf-performance profile**, replace that string with the correct performance profile name.

26.10.7. A test pod template for clusters that use OVS hardware offloading on OpenStack

The following **testpmd** pod demonstrates Open vSwitch (OVS) hardware offloading on Red Hat OpenStack Platform (RHOSP).

An example testpmd pod

```
apiVersion: v1
kind: Pod
metadata:
  name: testpmd-sriov
  namespace: mynamespace
  annotations:
    k8s.v1.cni.cncf.io/networks: hwoffload1
spec:
  runtimeClassName: performance-cnf-performanceprofile 1
  containers:
  - name: testpmd
    command: ["sleep", "99999"]
    image: registry.redhat.io/openshift4/dpdk-base-rhel8:v4.9
    securityContext:
      capabilities:
        add: ["IPC_LOCK","SYS_ADMIN"]
      privileged: true
      runAsUser: 0
    resources:
      requests:
        memory: 1000Mi
        hugepages-1Gi: 1Gi
        cpu: '2'
      limits:
        hugepages-1Gi: 1Gi
        cpu: '2'
        memory: 1000Mi
    volumeMounts:
    - mountPath: /mnt/huge
      name: hugepage
      readOnly: False
  volumes:
  - name: hugepage
    emptyDir:
      medium: HugePages
```

1 If your performance profile is not named **cnf-performance profile**, replace that string with the correct performance profile name.

26.10.8. Additional resources

- Creating a performance profile

- Reducing NIC queues using the Node Tuning Operator

- Provisioning a worker with real-time capabilities

- Installing the SR-IOV Network Operator

- Configuring an SR-IOV network device

- Dynamic IP address assignment configuration with Whereabouts

- Disabling interrupt processing for individual pods

- Configuring an SR-IOV Ethernet network attachment

- The app-netutil library provides several API methods for gathering network information about a container's parent pod.

26.11. USING POD-LEVEL BONDING

Bonding at the pod level is vital to enable workloads inside pods that require high availability and more throughput. With pod-level bonding, you can create a bond interface from multiple single root I/O virtualization (SR-IOV) virtual function interfaces in a kernel mode interface. The SR-IOV virtual functions are passed into the pod and attached to a kernel driver.

One scenario where pod level bonding is required is creating a bond interface from multiple SR-IOV virtual functions on different physical functions. Creating a bond interface from two different physical functions on the host can be used to achieve high availability and throughput at pod level.

For guidance on tasks such as creating a SR-IOV network, network policies, network attachment definitions and pods, see Configuring an SR-IOV network device .

26.11.1. Configuring a bond interface from two SR-IOV interfaces

Bonding enables multiple network interfaces to be aggregated into a single logical "bonded" interface. Bond Container Network Interface (Bond-CNI) brings bond capability into containers.

Bond-CNI can be created using Single Root I/O Virtualization (SR-IOV) virtual functions and placing them in the container network namespace.

OpenShift Container Platform only supports Bond-CNI using SR-IOV virtual functions. The SR-IOV Network Operator provides the SR-IOV CNI plugin needed to manage the virtual functions. Other CNIs or types of interfaces are not supported.

Prerequisites

- The SR-IOV Network Operator must be installed and configured to obtain virtual functions in a container.

- To configure SR-IOV interfaces, an SR-IOV network and policy must be created for each interface.

- The SR-IOV Network Operator creates a network attachment definition for each SR-IOV interface, based on the SR-IOV network and policy defined.

- The **linkState** is set to the default value **auto** for the SR-IOV virtual function.

26.11.1.1. Creating a bond network attachment definition

Now that the SR-IOV virtual functions are available, you can create a bond network attachment definition.

```
apiVersion: "k8s.cni.cncf.io/v1"
  kind: NetworkAttachmentDefinition
  metadata:
    name: bond-net1
    namespace: demo
  spec:
    config: '{
    "type": "bond", 1
    "cniVersion": "0.3.1",
    "name": "bond-net1",
    "mode": "active-backup", 2
    "failOverMac": 1, 3
    "linksInContainer": true, 4
    "miimon": "100",
    "mtu": 1500,
    "links": [ 5
        {"name": "net1"},
        {"name": "net2"}
     ],
    "ipam": {
        "type": "host-local",
        "subnet": "10.56.217.0/24",
        "routes": [{
        "dst": "0.0.0.0/0"
        }],
        "gateway": "10.56.217.1"
    }
    }'
```

[1] The cni-type is always set to **bond**.

[2] The **mode** attribute specifies the bonding mode.

> **NOTE**
>
> The bonding modes supported are:
>
> - **balance-rr** - 0
>
> - **active-backup** - 1
>
> - **balance-xor** - 2
>
> For **balance-rr** or **balance-xor** modes, you must set the **trust** mode to **on** for the SR-IOV virtual function.

[3] The **failover** attribute is mandatory for active-backup mode and must be set to 1.

[4] The **linksInContainer=true** flag informs the Bond CNI that the required interfaces are to be found inside the container. By default, Bond CNI looks for these interfaces on the host which does not work for integration with SRIOV and Multus.

[5] The **links** section defines which interfaces will be used to create the bond. By default, Multus names the attached interfaces as: "net", plus a consecutive number, starting with one.

26.11.1.2. Creating a pod using a bond interface

1. Test the setup by creating a pod with a YAML file named for example **podbonding.yaml** with content similar to the following:

```
apiVersion: v1
  kind: Pod
  metadata:
    name: bondpod1
    namespace: demo
    annotations:
      k8s.v1.cni.cncf.io/networks: demo/sriovnet1, demo/sriovnet2, demo/bond-net1 ❶
  spec:
    containers:
    - name: podexample
      image: quay.io/openshift/origin-network-interface-bond-cni:4.11.0
      command: ["/bin/bash", "-c", "sleep INF"]
```

❶ Note the network annotation: it contains two SR-IOV network attachments, and one bond network attachment. The bond attachment uses the two SR-IOV interfaces as bonded port interfaces.

2. Apply the yaml by running the following command:

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

3. Inspect the pod interfaces with the following command:

```
$ oc rsh -n demo bondpod1
sh-4.4#
sh-4.4# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
inet 127.0.0.1/8 scope host lo
valid_lft forever preferred_lft forever
3: eth0@if150: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1450 qdisc
noqueue state UP
link/ether 62:b1:b5:c8:fb:7a brd ff:ff:ff:ff:ff:ff
inet 10.244.1.122/24 brd 10.244.1.255 scope global eth0
valid_lft forever preferred_lft forever
4: net3: <BROADCAST,MULTICAST,UP,LOWER_UP400> mtu 1500 qdisc noqueue state
UP qlen 1000
link/ether 9e:23:69:42:fb:8a brd ff:ff:ff:ff:ff:ff ❶
inet 10.56.217.66/24 scope global bond0
valid_lft forever preferred_lft forever
43: net1: <BROADCAST,MULTICAST,UP,LOWER_UP800> mtu 1500 qdisc mq master
bond0 state UP qlen 1000
link/ether 9e:23:69:42:fb:8a brd ff:ff:ff:ff:ff:ff ❷
44: net2: <BROADCAST,MULTICAST,UP,LOWER_UP800> mtu 1500 qdisc mq master
bond0 state UP qlen 1000
link/ether 9e:23:69:42:fb:8a brd ff:ff:ff:ff:ff:ff ❸
```

❶ The bond interface is automatically named **net3**. To set a specific interface name add **@name** suffix to the pod's **k8s.v1.cni.cncf.io/networks** annotation.

2　The **net1** interface is based on an SR-IOV virtual function.

3　The **net2** interface is based on an SR-IOV virtual function.

> **NOTE**
>
> If no interface names are configured in the pod annotation, interface names are assigned automatically as **net\<n\>**, with **\<n\>** starting at **1**.

4. Optional: If you want to set a specific interface name for example **bond0**, edit the **k8s.v1.cni.cncf.io/networks** annotation and set **bond0** as the interface name as follows:

```
annotations:
    k8s.v1.cni.cncf.io/networks: demo/sriovnet1, demo/sriovnet2, demo/bond-net1@bond0
```

26.12. CONFIGURING HARDWARE OFFLOADING

As a cluster administrator, you can configure hardware offloading on compatible nodes to increase data processing performance and reduce load on host CPUs.

26.12.1. About hardware offloading

Open vSwitch hardware offloading is a method of processing network tasks by diverting them away from the CPU and offloading them to a dedicated processor on a network interface controller. As a result, clusters can benefit from faster data transfer speeds, reduced CPU workloads, and lower computing costs.

The key element for this feature is a modern class of network interface controllers known as SmartNICs. A SmartNIC is a network interface controller that is able to handle computationally-heavy network processing tasks. In the same way that a dedicated graphics card can improve graphics performance, a SmartNIC can improve network performance. In each case, a dedicated processor improves performance for a specific type of processing task.

In OpenShift Container Platform, you can configure hardware offloading for bare metal nodes that have a compatible SmartNIC. Hardware offloading is configured and enabled by the SR-IOV Network Operator.

Hardware offloading is not compatible with all workloads or application types. Only the following two communication types are supported:

- pod-to-pod

- pod-to-service, where the service is a ClusterIP service backed by a regular pod

In all cases, hardware offloading takes place only when those pods and services are assigned to nodes that have a compatible SmartNIC. Suppose, for example, that a pod on a node with hardware offloading tries to communicate with a service on a regular node. On the regular node, all the processing takes place in the kernel, so the overall performance of the pod-to-service communication is limited to the maximum performance of that regular node. Hardware offloading is not compatible with DPDK applications.

Enabling hardware offloading on a node, but not configuring pods to use, it can result in decreased throughput performance for pod traffic. You cannot configure hardware offloading for pods that are managed by OpenShift Container Platform.

26.12.2. Supported devices

Hardware offloading is supported on the following network interface controllers:

Table 26.15. Supported network interface controllers

Manufacturer	Model	Vendor ID	Device ID
Mellanox	MT27800 Family [ConnectX-5]	15b3	1017
Mellanox	MT28880 Family [ConnectX-5 Ex]	15b3	1019
Mellanox	MT2892 Family [ConnectX-6 Dx]	15b3	101d
Mellanox	MT2894 Family [ConnectX-6 Lx]	15b3	101f
Mellanox	MT42822 BlueField-2 in ConnectX-6 NIC mode	15b3	a2d6

26.12.3. Prerequisites

- Your cluster has at least one bare metal machine with a network interface controller that is supported for hardware offloading.

- You installed the SR-IOV Network Operator.

- Your cluster uses the OVN-Kubernetes network plugin.

- In your OVN-Kubernetes network plugin configuration, the **gatewayConfig.routingViaHost** field is set to **false**.

26.12.4. Configuring a machine config pool for hardware offloading

To enable hardware offloading, you must first create a dedicated machine config pool and configure it to work with the SR-IOV Network Operator.

Prerequisites

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

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

Procedure

1. Create a machine config pool for machines you want to use hardware offloading on.

   a. Create a file, such as **mcp-offloading.yaml**, with content like the following example:

   ```
   apiVersion: machineconfiguration.openshift.io/v1
   kind: MachineConfigPool
   metadata:
     name: mcp-offloading ❶
   spec:
   ```

```
      machineConfigSelector:
        matchExpressions:
          - {key: machineconfiguration.openshift.io/role, operator: In, values: [worker,mcp-
      offloading]} 2
        nodeSelector:
          matchLabels:
            node-role.kubernetes.io/mcp-offloading: "" 3
```

1 2 The name of your machine config pool for hardware offloading.

3 This node role label is used to add nodes to the machine config pool.

b. Apply the configuration for the machine config pool:

```
$ oc create -f mcp-offloading.yaml
```

2. Add nodes to the machine config pool. Label each node with the node role label of your pool:

```
$ oc label node worker-2 node-role.kubernetes.io/mcp-offloading=""
```

3. Optional: To verify that the new pool is created, run the following command:

```
$ oc get nodes
```

Example output

```
NAME      STATUS  ROLES                AGE  VERSION
master-0  Ready   master               2d   v1.26.0
master-1  Ready   master               2d   v1.26.0
master-2  Ready   master               2d   v1.26.0
worker-0  Ready   worker               2d   v1.26.0
worker-1  Ready   worker               2d   v1.26.0
worker-2  Ready   mcp-offloading,worker  47h  v1.26.0
worker-3  Ready   mcp-offloading,worker  47h  v1.26.0
```

4. Add this machine config pool to the **SriovNetworkPoolConfig** custom resource:

a. Create a file, such as **sriov-pool-config.yaml**, with content like the following example:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkPoolConfig
metadata:
  name: sriovnetworkpoolconfig-offload
  namespace: openshift-sriov-network-operator
spec:
  ovsHardwareOffloadConfig:
    name: mcp-offloading 1
```

1 The name of your machine config pool for hardware offloading.

b. Apply the configuration:

```
$ oc create -f <SriovNetworkPoolConfig_name>.yaml
```
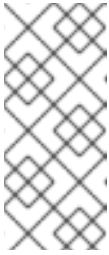
> **NOTE**
>
> When you apply the configuration specified in a **SriovNetworkPoolConfig**
> object, the SR-IOV Operator drains and restarts the nodes in the machine
> config pool.
>
> It might take several minutes for a configuration changes to apply.

26.12.5. Configuring the SR-IOV network node policy

You can create an SR-IOV network device configuration for a node by creating an SR-IOV network node policy. To enable hardware offloading, you must define the **.spec.eSwitchMode** field with the value **"switchdev"**.

The following procedure creates an SR-IOV interface for a network interface controller with hardware offloading.

Prerequisites

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

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

Procedure

1. Create a file, such as **sriov-node-policy.yaml**, with content like the following example:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: sriov-node-policy <.>
  namespace: openshift-sriov-network-operator
spec:
  deviceType: netdevice <.>
  eSwitchMode: "switchdev" <.>
  nicSelector:
    deviceID: "1019"
    rootDevices:
    - 0000:d8:00.0
    vendor: "15b3"
    pfNames:
    - ens8f0
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  numVfs: 6
  priority: 5
  resourceName: mlxnics
```

<.> The name for the custom resource object. <.> Required. Hardware offloading is not supported with **vfio-pci**. <.> Required.

2. Apply the configuration for the policy:

```
$ oc create -f sriov-node-policy.yaml
```

> **NOTE**
>
> When you apply the configuration specified in a **SriovNetworkPoolConfig** object, the SR-IOV Operator drains and restarts the nodes in the machine config pool.
>
> It might take several minutes for a configuration change to apply.

26.12.5.1. An example SR-IOV network node policy for OpenStack

The following example describes an SR-IOV interface for a network interface controller (NIC) with hardware offloading on Red Hat OpenStack Platform (RHOSP).

An SR-IOV interface for a NIC with hardware offloading on RHOSP

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: ${name}
  namespace: openshift-sriov-network-operator
spec:
  deviceType: switchdev
  isRdma: true
  nicSelector:
    netFilter: openstack/NetworkID:${net_id}
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: 'true'
  numVfs: 1
  priority: 99
  resourceName: ${name}
```

26.12.6. Creating a network attachment definition

After you define the machine config pool and the SR-IOV network node policy, you can create a network attachment definition for the network interface card you specified.

Prerequisites

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

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

Procedure

1. Create a file, such as **net-attach-def.yaml**, with content like the following example:

   ```
   apiVersion: "k8s.cni.cncf.io/v1"
   kind: NetworkAttachmentDefinition
   metadata:
     name: net-attach-def <.>
     namespace: net-attach-def <.>
   ```

```
    annotations:
      k8s.v1.cni.cncf.io/resourceName: openshift.io/mlxnics <.>
  spec:
    config: '{"cniVersion":"0.3.1","name":"ovn-kubernetes","type":"ovn-k8s-cni-overlay","ipam":
    {},"dns":{}}'
```

<.> The name for your network attachment definition. <.> The namespace for your network attachment definition. <.> This is the value of the **spec.resourceName** field you specified in the **SriovNetworkNodePolicy** object.

2. Apply the configuration for the network attachment definition:

```
$ oc create -f net-attach-def.yaml
```

Verification

- Run the following command to see whether the new definition is present:

```
$ oc get net-attach-def -A
```

Example output

```
NAMESPACE        NAME           AGE
net-attach-def   net-attach-def  43h
```

26.12.7. Adding the network attachment definition to your pods

After you create the machine config pool, the **SriovNetworkPoolConfig** and **SriovNetworkNodePolicy** custom resources, and the network attachment definition, you can apply these configurations to your pods by adding the network attachment definition to your pod specifications.

Procedure

- In the pod specification, add the **.metadata.annotations.k8s.v1.cni.cncf.io/networks** field and specify the network attachment definition you created for hardware offloading:

```
....
metadata:
  annotations:
    v1.multus-cni.io/default-network: net-attach-def/net-attach-def <.>
```

<.> The value must be the name and namespace of the network attachment definition you created for hardware offloading.

26.13. SWITCHING BLUEFIELD-2 FROM DPU TO NIC

You can switch the Bluefield-2 network device from data processing unit (DPU) mode to network interface controller (NIC) mode.

26.13.1. Switching Bluefield-2 from DPU mode to NIC mode

Use the following procedure to switch Bluefield-2 from data processing units (DPU) mode to network interface controller (NIC) mode.

> **IMPORTANT**
>
> Currently, only switching Bluefield-2 from DPU to NIC mode is supported. Switching from NIC mode to DPU mode is unsupported.

Prerequisites

- You have installed the SR-IOV Network Operator. For more information, see "Installing SR-IOV Network Operator".

- You have updated Bluefield-2 to the latest firmware. For more information, see Firmware for NVIDIA BlueField-2.

Procedure

1. Add the following labels to each of your worker nodes by entering the following commands:

   ```
   $ oc label node <example_node_name_one> node-role.kubernetes.io/sriov=
   ```

   ```
   $ oc label node <example_node_name_two> node-role.kubernetes.io/sriov=
   ```

2. Create a machine config pool for the SR-IOV Network Operator, for example:

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

3. Apply the following **machineconfig.yaml** file to the worker nodes:

   ```
   apiVersion: machineconfiguration.openshift.io/v1
   kind: MachineConfig
   metadata:
     labels:
       machineconfiguration.openshift.io/role: sriov
     name: 99-bf2-dpu
   spec:
     config:
       ignition:
         version: 3.2.0
       storage:
         files:
         - contents:
             source: data:text/plain;charset=utf-
   ```

8;base64,ZmluZF9jb250YWluZXIoKSB7CiAgY3JpY3RsIHBzIC1vIGpzb24gfCBqcSAtciAnLmNv
bnRhaW5lcnNbXSB8IHNlbGVjdCgubWV0YWRhdGEubmFtZT09InNyaW92LW5ldHdvcmstY29
uZmlnLWRhZW1vbiIpIHwgLmlkJwp9CnVudGlsIG91dHB1dD0kKGZpbmRfY29udGFpbmVyKT
sgW1sgLW4gIiRvdXRwdXQiIF1dOyBkbwogIGVjaG8gIndhaXRpbmcgZm9yIGNvbnRhaW5lciB
0byBjb21lIHVwIgogIHNsZWVwIDE7CmRvbmUKISBzdWRvIGNyaWN0bCBleGVjICRvdXRwdX
QgL2JpbmRhdGEvc2NyaXB0cy9iZjItc3dpdGNoLW1vZGUuc2ggIiRAIgo=

      mode: 0755
      overwrite: true
      path: /etc/default/switch_in_sriov_config_daemon.sh
    systemd:
     units:
     - name: dpu-switch.service
      enabled: true
      contents: |
       [Unit]
       Description=Switch BlueField2 card to NIC/DPU mode
       RequiresMountsFor=%t/containers
       Wants=network.target
       After=network-online.target kubelet.service
       [Service]
       SuccessExitStatus=0 120
       RemainAfterExit=True
       ExecStart=/bin/bash -c '/etc/default/switch_in_sriov_config_daemon.sh nic || shutdown
-r now' **1**
       Type=oneshot
       [Install]
       WantedBy=multi-user.target

1   Optional: The PCI address of a specific card can optionally be specified, for example **ExecStart=/bin/bash -c '/etc/default/switch_in_sriov_config_daemon.sh nic 0000:5e:00.0 || echo done'**. By default, the first device is selected. If there is more than one device, you must specify which PCI address to be used. The PCI address must be the same on all nodes that are switching Bluefield-2 from DPU mode to NIC mode.

4. Wait for the worker nodes to restart. After restarting, the Bluefield-2 network device on the worker nodes is switched into NIC mode.

Additional resources

Installing SR-IOV Network Operator

26.14. UNINSTALLING THE SR-IOV NETWORK OPERATOR

To uninstall the SR-IOV Network Operator, you must delete any running SR-IOV workloads, uninstall the Operator, and delete the webhooks that the Operator used.

26.14.1. Uninstalling the SR-IOV Network Operator

As a cluster administrator, you can uninstall the SR-IOV Network Operator.

Prerequisites

- You have access to an OpenShift Container Platform cluster using an account with **cluster-admin** permissions.

- You have the SR-IOV Network Operator installed.

Procedure

1. Delete all SR-IOV custom resources (CRs):

   ```
   $ oc delete sriovnetwork -n openshift-sriov-network-operator --all
   ```

   ```
   $ oc delete sriovnetworknodepolicy -n openshift-sriov-network-operator --all
   ```

   ```
   $ oc delete sriovibnetwork -n openshift-sriov-network-operator --all
   ```

2. Follow the instructions in the "Deleting Operators from a cluster" section to remove the SR-IOV Network Operator from your cluster.

3. Delete the SR-IOV custom resource definitions that remain in the cluster after the SR-IOV Network Operator is uninstalled:

   ```
   $ oc delete crd sriovibnetworks.sriovnetwork.openshift.io
   ```

   ```
   $ oc delete crd sriovnetworknodepolicies.sriovnetwork.openshift.io
   ```

   ```
   $ oc delete crd sriovnetworknodestates.sriovnetwork.openshift.io
   ```

   ```
   $ oc delete crd sriovnetworkpoolconfigs.sriovnetwork.openshift.io
   ```

   ```
   $ oc delete crd sriovnetworks.sriovnetwork.openshift.io
   ```

   ```
   $ oc delete crd sriovoperatorconfigs.sriovnetwork.openshift.io
   ```

4. Delete the SR-IOV webhooks:

   ```
   $ oc delete mutatingwebhookconfigurations network-resources-injector-config
   ```

   ```
   $ oc delete MutatingWebhookConfiguration sriov-operator-webhook-config
   ```

   ```
   $ oc delete ValidatingWebhookConfiguration sriov-operator-webhook-config
   ```

5. Delete the SR-IOV Network Operator namespace:

   ```
   $ oc delete namespace openshift-sriov-network-operator
   ```

Additional resources

- [Deleting Operators from a cluster](#)

CHAPTER 27. OVN-KUBERNETES NETWORK PLUGIN

27.1. ABOUT THE OVN-KUBERNETES NETWORK PLUGIN

The OpenShift Container Platform cluster uses a virtualized network for pod and service networks.

Part of Red Hat OpenShift Networking, the OVN-Kubernetes network plugin is the default network provider for OpenShift Container Platform. OVN-Kubernetes is based on Open Virtual Network (OVN) and provides an overlay-based networking implementation. A cluster that uses the OVN-Kubernetes plugin also runs Open vSwitch (OVS) on each node. OVN configures OVS on each node to implement the declared network configuration.
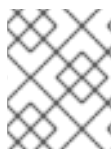
> **NOTE**
>
> OVN-Kubernetes is the default networking solution for OpenShift Container Platform and single-node OpenShift deployments.

OVN-Kubernetes, which arose from the OVS project, uses many of the same constructs, such as open flow rules, to determine how packets travel through the network. For more information, see the Open Virtual Network website.

OVN-Kubernetes is a series of daemons for OVS that translate virtual network configurations into **OpenFlow** rules. **OpenFlow** is a protocol for communicating with network switches and routers, providing a means for remotely controlling the flow of network traffic on a network device, allowing network administrators to configure, manage, and monitor the flow of network traffic.

OVN-Kubernetes provides more of the advanced functionality not available with **OpenFlow**. OVN supports distributed virtual routing, distributed logical switches, access control, DHCP and DNS. OVN implements distributed virtual routing within logic flows which equate to open flows. So for example if you have a pod that sends out a DHCP request on the network, it sends out that broadcast looking for DHCP address there will be a logic flow rule that matches that packet, and it responds giving it a gateway, a DNS server an IP address and so on.

OVN-Kubernetes runs a daemon on each node. There are daemon sets for the databases and for the OVN controller that run on every node. The OVN controller programs the Open vSwitch daemon on the nodes to support the network provider features; egress IPs, firewalls, routers, hybrid networking, IPSEC encryption, IPv6, network policy, network policy logs, hardware offloading and multicast.

27.1.1. OVN-Kubernetes purpose

The OVN-Kubernetes network plugin is an open-source, fully-featured Kubernetes CNI plugin that uses Open Virtual Network (OVN) to manage network traffic flows. OVN is a community developed, vendor-agnostic network virtualization solution. The OVN-Kubernetes network plugin:

- Uses OVN (Open Virtual Network) to manage network traffic flows. OVN is a community developed, vendor-agnostic network virtualization solution.

- Implements Kubernetes network policy support, including ingress and egress rules.

- Uses the Geneve (Generic Network Virtualization Encapsulation) protocol rather than VXLAN to create an overlay network between nodes.

The OVN-Kubernetes network plugin provides the following advantages over OpenShift SDN.

- Full support for IPv6 single-stack and IPv4/IPv6 dual-stack networking on supported platforms

- Support for hybrid clusters with both Linux and Microsoft Windows workloads

- Optional IPsec encryption of intra-cluster communications

- Offload of network data processing from host CPU to compatible network cards and data processing units (DPUs)

27.1.2. Supported network plugin feature matrix

Red Hat OpenShift Networking offers two options for the network plugin, OpenShift SDN and OVN-Kubernetes, for the network plugin. The following table summarizes the current feature support for both network plugins:

Table 27.1. Default CNI network plugin feature comparison

Feature	OVN-Kubernetes	OpenShift SDN
Egress IPs	Supported	Supported
Egress firewall [1]	Supported	Supported
Egress router	Supported [2]	Supported
Hybrid networking	Supported	Not supported
IPsec encryption for intra-cluster communication	Supported	Not supported
IPv6	Supported [3] [4]	Not supported
Kubernetes network policy	Supported	Supported
Kubernetes network policy logs	Supported	Not supported
Hardware offloading	Supported	Not supported
Multicast	Supported	Supported

1. Egress firewall is also known as egress network policy in OpenShift SDN. This is not the same as network policy egress.

2. Egress router for OVN-Kubernetes supports only redirect mode.

3. IPv6 is supported only on bare metal, IBM Power, and IBM Z clusters.

4. IPv6 single stack does not support Kubernetes NMState and is not supported on IBM Power and IBM Z clusters.

27.1.3. OVN-Kubernetes IPv6 and dual-stack limitations

The OVN-Kubernetes network plugin has the following limitations:

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

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

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

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

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

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

27.1.4. Session affinity

Session affinity is a feature that applies to Kubernetes **Service** objects. You can use *session affinity* if you want to ensure that each time you connect to a <service_VIP>:<Port>, the traffic is always load balanced to the same back end. For more information, including how to set session affinity based on a client's IP address, see Session affinity.

Stickiness timeout for session affinity
The OVN-Kubernetes network plugin for OpenShift Container Platform calculates the stickiness timeout for a session from a client based on the last packet. For example, if you run a **curl** command 10 times, the sticky session timer starts from the tenth packet not the first. As a result, if the client is continuously contacting the service, then the session never times out. The timeout starts when the service has not received a packet for the amount of time set by the **timeoutSeconds** parameter.

Additional resources

- Configuring an egress firewall for a project

- About network policy

- Logging network policy events

- [Enabling multicast for a project](#)

- [Configuring IPsec encryption](#)

- [Network [operator.openshift.io/v1]](#)

27.2. OVN-KUBERNETES ARCHITECTURE

27.2.1. Introduction to OVN-Kubernetes architecture

The following diagram shows the OVN-Kubernetes architecture.

Figure 27.1. OVK-Kubernetes architecture
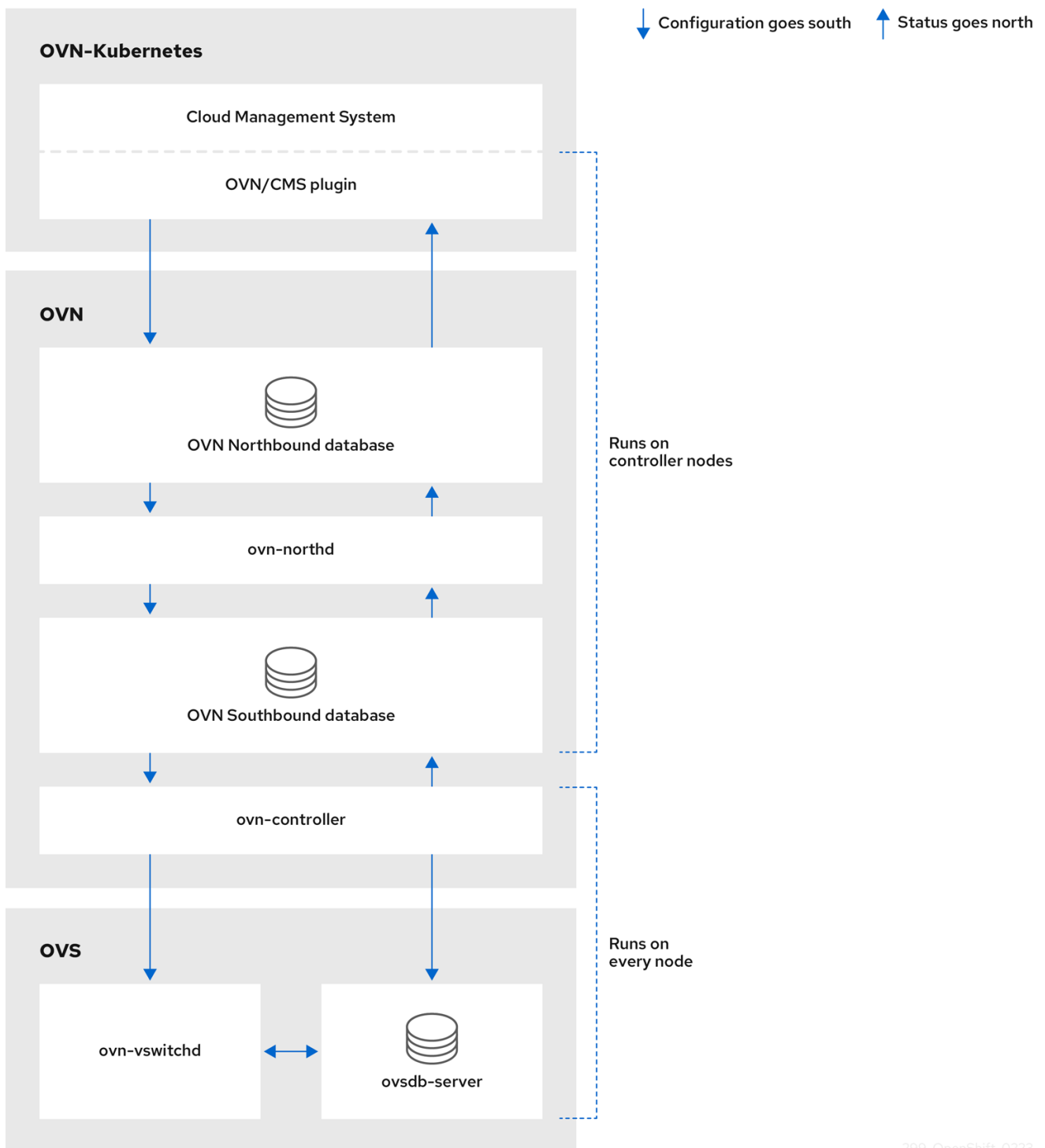
The key components are:

- **Cloud Management System (CMS)** – A platform specific client for OVN that provides a CMS specific plugin for OVN integration. The plugin translates the cloud management system's concept of the logical network configuration, stored in the CMS configuration database in a CMS-specific format, into an intermediate representation understood by OVN.

- **OVN Northbound database (nbdb)** - Stores the logical network configuration passed by the CMS plugin.

- **OVN Southbound database (sbdb)** – Stores the physical and logical network configuration state for OpenVswitch (OVS) system on each node, including tables that bind them.

- **ovn-northd** – This is the intermediary client between **nbdb** and **sbdb**. It translates the logical network configuration in terms of conventional network concepts, taken from the **nbdb**, into logical data path flows in the **sbdb** below it. The container name is **northd** and it runs in the **ovnkube-master** pods.

- **ovn-controller** – This is the OVN agent that interacts with OVS and hypervisors, for any information or update that is needed for **sbdb**. The **ovn-controller** reads logical flows from the **sbdb**, translates them into **OpenFlow** flows and sends them to the node's OVS daemon. The container name is **ovn-controller** and it runs in the **ovnkube-node** pods.

The OVN northbound database has the logical network configuration passed down to it by the cloud management system (CMS). The OVN northbound Database contains the current desired state of the network, presented as a collection of logical ports, logical switches, logical routers, and more. The **ovn-northd** (**northd** container) connects to the OVN northbound database and the OVN southbound database. It translates the logical network configuration in terms of conventional network concepts, taken from the OVN northbound Database, into logical data path flows in the OVN southbound database.

The OVN southbound database has physical and logical representations of the network and binding tables that link them together. Every node in the cluster is represented in the southbound database, and you can see the ports that are connected to it. It also contains all the logic flows, the logic flows are shared with the **ovn-controller** process that runs on each node and the **ovn-controller** turns those into **OpenFlow** rules to program **Open vSwitch**.

The Kubernetes control plane nodes each contain an **ovnkube-master** pod which hosts containers for the OVN northbound and southbound databases. All OVN northbound databases form a **Raft** cluster and all southbound databases form a separate **Raft** cluster. At any given time a single **ovnkube-master** is the leader and the other **ovnkube-master** pods are followers.

27.2.2. Listing all resources in the OVN-Kubernetes project

Finding the resources and containers that run in the OVN-Kubernetes project is important to help you understand the OVN-Kubernetes networking implementation.

Prerequisites

- Access to the cluster as a user with the **cluster-admin** role.

- The OpenShift CLI (**oc**) installed.

Procedure

1. Run the following command to get all resources, endpoints, and **ConfigMaps** in the OVN-Kubernetes project:

   ```
   $ oc get all,ep,cm -n openshift-ovn-kubernetes
   ```

   Example output

   ```
   NAME                      READY   STATUS    RESTARTS      AGE
   pod/ovnkube-master-9g7zt   6/6    Running   1 (48m ago)   57m
   pod/ovnkube-master-lqs4v   6/6    Running   0             57m
   pod/ovnkube-master-vxhtq   6/6    Running   0             57m
   pod/ovnkube-node-9k9kc     5/5    Running   0             57m
   pod/ovnkube-node-jg52r     5/5    Running   0             51m
   ```

```
pod/ovnkube-node-k8wf7     5/5     Running   0          57m
pod/ovnkube-node-tlwk6     5/5     Running   0          47m
pod/ovnkube-node-xsvnk     5/5     Running   0          57m


NAME                        TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
service/ovn-kubernetes-master   ClusterIP   None         <none>        9102/TCP         57m
service/ovn-kubernetes-node     ClusterIP   None         <none>        9103/TCP,9105/TCP
57m
service/ovnkube-db              ClusterIP   None         <none>        9641/TCP,9642/TCP   57m


NAME                        DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE
NODE SELECTOR                            AGE
daemonset.apps/ovnkube-master   3         3         3       3            3
beta.kubernetes.io/os=linux,node-role.kubernetes.io/master=   57m
daemonset.apps/ovnkube-node     5         5         5       5            5
beta.kubernetes.io/os=linux                  57m


NAME                        ENDPOINTS                                           AGE
endpoints/ovn-kubernetes-master   10.0.132.11:9102,10.0.151.18:9102,10.0.192.45:9102
57m
endpoints/ovn-kubernetes-node     10.0.132.11:9105,10.0.143.72:9105,10.0.151.18:9105 + 7
more...   57m
endpoints/ovnkube-db              10.0.132.11:9642,10.0.151.18:9642,10.0.192.45:9642 + 3
more...   57m


NAME                        DATA   AGE
configmap/control-plane-status     1      55m
configmap/kube-root-ca.crt         1      57m
configmap/openshift-service-ca.crt   1      57m
configmap/ovn-ca                   1      57m
configmap/ovnkube-config           1      57m
configmap/signer-ca                1      57m
```

There are three **ovnkube-masters** that run on the control plane nodes, and two daemon sets used to deploy the **ovnkube-master** and **ovnkube-node** pods. There is one **ovnkube-node** pod for each node in the cluster. The **ovnkube-config ConfigMap** has the OpenShift Container Platform OVN-Kubernetes configurations started by online-master and **ovnkube-node**.

2. List all the containers in the **ovnkube-master** pods by running the following command:

```
$ oc get pods ovnkube-master-9g7zt \
-o jsonpath='{.spec.containers[*].name}' -n openshift-ovn-kubernetes
```

Expected output

```
northd nbdb kube-rbac-proxy sbdb ovnkube-master ovn-dbchecker
```

The **ovnkube-master** pod is made up of several containers. It is responsible for hosting the northbound database (**nbdb** container), the southbound database (**sbdb** container), watching for cluster events for pods, egressIP, namespaces, services, endpoints, egress firewall, and network policy and writing them to the northbound database (**ovnkube-master** pod), as well as managing pod subnet allocation to nodes.

3. List all the containers in the **ovnkube-node** pods by running the following command:

```
$ oc get pods ovnkube-node-jg52r \
-o jsonpath='{.spec.containers[*].name}' -n openshift-ovn-kubernetes
```

Expected output

```
ovn-controller ovn-acl-logging kube-rbac-proxy kube-rbac-proxy-ovn-metrics ovnkube-node
```

The **ovnkube-node** pod has a container (**ovn-controller**) that resides on each OpenShift Container Platform node. Each node's **ovn-controller** connects the OVN northbound to the OVN southbound database to learn about the OVN configuration. The **ovn-controller** connects southbound to **ovs-vswitchd** as an OpenFlow controller, for control over network traffic, and to the local **ovsdb-server** to allow it to monitor and control Open vSwitch configuration.

4. List the currently elected OVN-Kubernetes master leader by running the following command:

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

   ### Expected output

   ```
   NAME                 HOLDER                         AGE
   ovn-kubernetes-master   ci-ln-gz990pb-72292-rthz2-master-2   50m
   ```

27.2.3. Listing the OVN-Kubernetes northbound database contents

To understand logic flow rules you need to examine the northbound database and understand what objects are there to see how they are translated into logic flow rules. The up to date information is present on the OVN Raft leader and this procedure describes how to find the Raft leader and subsequently query it to list the OVN northbound database contents.

Prerequisites

- Access to the cluster as a user with the **cluster-admin** role.

- The OpenShift CLI (**oc**) installed.

Procedure

1. Find the OVN Raft leader for the northbound database.

   > **NOTE**
   >
   > The Raft leader stores the most up to date information.

   a. List the pods by running the following command:

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

      ### Example output

      ```
      NAME                 READY  STATUS   RESTARTS      AGE
      ovnkube-master-7j97q  6/6    Running  2 (148m ago)  149m
      ovnkube-master-gt4ms  6/6    Running  1 (140m ago)  147m
      ```

```
ovnkube-master-mk6p6   6/6     Running   0              148m
ovnkube-node-8qvtr     5/5     Running   0              149m
ovnkube-node-fqdc9     5/5     Running   0              149m
ovnkube-node-tlfwv     5/5     Running   0              149m
ovnkube-node-wlwkn     5/5     Running   0              142m
```

b. Choose one of the master pods at random and run the following command:

```
$ oc exec -n openshift-ovn-kubernetes ovnkube-master-7j97q \
-- /usr/bin/ovn-appctl -t /var/run/ovn/ovnnb_db.ctl \
--timeout=3 cluster/status OVN_Northbound
```

Example output

```
Defaulted container "northd" out of: northd, nbdb, kube-rbac-proxy, sbdb, ovnkube-
master, ovn-dbchecker
1c57
Name: OVN_Northbound
Cluster ID: c48a (c48aa5c0-a704-4c77-a066-24fe99d9b338)
Server ID: 1c57 (1c57b6fc-2849-49b7-8679-fbf18bafe339)
Address: ssl:10.0.147.219:9643
Status: cluster member
Role: follower   ❶
Term: 5
Leader: 2b4f   ❷
Vote: unknown

Election timer: 10000
Log: [2, 3018]
Entries not yet committed: 0
Entries not yet applied: 0
Connections: ->0000 ->0000 <-8844 <-2b4f
Disconnections: 0
Servers:
    1c57 (1c57 at ssl:10.0.147.219:9643) (self)
    8844 (8844 at ssl:10.0.163.212:9643) last msg 8928047 ms ago
    2b4f (2b4f at ssl:10.0.242.240:9643) last msg 620 ms ago   ❸
```

❶ This pod is identified as a follower

❷ The leader is identified as **2b4f**

❸ The **2b4f** is on IP address **10.0.242.240**

c. Find the **ovnkube-master** pod running on IP Address **10.0.242.240** using the following command:

```
$ oc get po -o wide -n openshift-ovn-kubernetes | grep 10.0.242.240 | grep -v ovnkube-
node
```

Example output

```
ovnkube-master-gt4ms  6/6     Running       1 (143m ago)  150m  10.0.242.240  ip-
10-0-242-240.ec2.internal  <none>         <none>
```

The **ovnkube-master-gt4ms** pod runs on IP Address 10.0.242.240.

2. Run the following command to show all the objects in the northbound database:

```
$ oc exec -n openshift-ovn-kubernetes -it ovnkube-master-gt4ms \
-c northd -- ovn-nbctl show
```

The output is too long to list here. The list includes the NAT rules, logical switches, load balancers and so on.

Run the following command to display the options available with the command **ovn-nbctl**:

```
$ oc exec -n openshift-ovn-kubernetes -it ovnkube-master-mk6p6 \
-c northd ovn-nbctl --help
```

You can narrow down and focus on specific components by using some of the following commands:

3. Run the following command to show the list of logical routers:

```
$ oc exec -n openshift-ovn-kubernetes -it ovnkube-master-gt4ms \
-c northd -- ovn-nbctl lr-list
```

Example output

```
f971f1f3-5112-402f-9d1e-48f1d091ff04 (GR_ip-10-0-145-205.ec2.internal)
69c992d8-a4cf-429e-81a3-5361209ffe44 (GR_ip-10-0-147-219.ec2.internal)
7d164271-af9e-4283-b84a-48f2a44851cd (GR_ip-10-0-163-212.ec2.internal)
111052e3-c395-408b-97b2-8dd0a20a29a5 (GR_ip-10-0-165-9.ec2.internal)
ed50ce33-df5d-48e8-8862-2df6a59169a0 (GR_ip-10-0-209-170.ec2.internal)
f44e2a96-8d1e-4a4d-abae-ed8728ac6851 (GR_ip-10-0-242-240.ec2.internal)
ef3d0057-e557-4b1a-b3c6-fcc3463790b0 (ovn_cluster_router)
```

> **NOTE**
>
> From this output you can see there is router on each node plus an **ovn_cluster_router**.

4. Run the following command to show the list of logical switches:

```
$ oc exec -n openshift-ovn-kubernetes -it ovnkube-master-gt4ms \
-c northd -- ovn-nbctl ls-list
```

Example output

```
82808c5c-b3bc-414a-bb59-8fec4b07eb14 (ext_ip-10-0-145-205.ec2.internal)
3d22444f-0272-4c51-afc6-de9e03db3291 (ext_ip-10-0-147-219.ec2.internal)
bf73b9df-59ab-4c58-a456-ce8205b34ac5 (ext_ip-10-0-163-212.ec2.internal)
bee1e8d0-ec87-45eb-b98b-63f9ec213e5e (ext_ip-10-0-165-9.ec2.internal)
```

```
812f08f2-6476-4abf-9a78-635f8516f95e (ext_ip-10-0-209-170.ec2.internal)
f65e710b-32f9-482b-8eab-8d96a44799c1 (ext_ip-10-0-242-240.ec2.internal)
84dad700-afb8-4129-86f9-923a1ddeace9 (ip-10-0-145-205.ec2.internal)
1b7b448b-e36c-4ca3-9f38-4a2cf6814bfd (ip-10-0-147-219.ec2.internal)
d92d1f56-2606-4f23-8b6a-4396a78951de (ip-10-0-163-212.ec2.internal)
6864a6b2-de15-4de3-92d8-f95014b6f28f (ip-10-0-165-9.ec2.internal)
c26bf618-4d7e-4afd-804f-1a2cbc96ec6d (ip-10-0-209-170.ec2.internal)
ab9a4526-44ed-4f82-ae1c-e20da04947d9 (ip-10-0-242-240.ec2.internal)
a8588aba-21da-4276-ba0f-9d68e88911f0 (join)
```
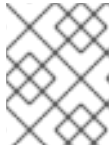
> **NOTE**
>
> From this output you can see there is an ext switch for each node plus switches with the node name itself and a join switch.

5. Run the following command to show the list of load balancers:

```
$ oc exec -n openshift-ovn-kubernetes -it ovnkube-master-gt4ms \
-c northd -- ovn-nbctl lb-list
```

Example output

```
UUID                                    LB              PROTO   VIP            IPs
f0fb50f9-4968-4b55-908c-616bae4db0a2    Service_default/    tcp        172.30.0.1:443
10.0.147.219:6443,10.0.163.212:6443,169.254.169.2:6443
0dc42012-4f5b-432e-ae01-2cc4bfe81b00    Service_default/    tcp        172.30.0.1:443
10.0.147.219:6443,169.254.169.2:6443,10.0.242.240:6443
f7fff5d5-5eff-4a40-98b1-3a4ba8f7f69c    Service_default/  tcp       172.30.0.1:443
169.254.169.2:6443,10.0.163.212:6443,10.0.242.240:6443
12fe57a0-50a4-4a1b-ac10-5f288badee07    Service_default/    tcp       172.30.0.1:443
10.0.147.219:6443,10.0.163.212:6443,10.0.242.240:6443
3f137fbf-0b78-4875-ba44-fbf89f254cf7    Service_openshif    tcp       172.30.23.153:443
10.130.0.14:8443
174199fe-0562-4141-b410-12094db922a7    Service_openshif    tcp       172.30.69.51:50051
10.130.0.84:50051
5ee2d4bd-c9e2-4d16-a6df-f54cd17c9ac3    Service_openshif    tcp       172.30.143.87:9001
10.0.145.205:9001,10.0.147.219:9001,10.0.163.212:9001,10.0.165.9:9001,10.0.209.170:9001,
10.0.242.240:9001
a056ae3d-83f8-45bc-9c80-ef89bce7b162    Service_openshif    tcp       172.30.164.74:443
10.0.147.219:6443,10.0.163.212:6443,10.0.242.240:6443
bac51f3d-9a6f-4f5e-ac02-28fd343a332a    Service_openshif    tcp       172.30.0.10:53
10.131.0.6:5353
                                        tcp        172.30.0.10:9154       10.131.0.6:9154
48105bbc-51d7-4178-b975-417433f9c20a    Service_openshif    tcp       172.30.26.159:2379
10.0.147.219:2379,169.254.169.2:2379,10.0.242.240:2379
                                        tcp        172.30.26.159:9979
10.0.147.219:9979,169.254.169.2:9979,10.0.242.240:9979
7de2b8fc-342a-415f-ac13-1a493f4e39c0    Service_openshif    tcp       172.30.53.219:443
10.128.0.7:8443
                                        tcp        172.30.53.219:9192     10.128.0.7:9192
2cef36bc-d720-4afb-8d95-9350eff1d27a    Service_openshif    tcp       172.30.81.66:443
10.128.0.23:8443
365cb6fb-e15e-45a4-a55b-21868b3cf513    Service_openshif    tcp       172.30.96.51:50051
10.130.0.19:50051
```

```
41691cbb-ec55-4cdb-8431-afce679c5e8d    Service_openshif    tcp       172.30.98.218:9099
169.254.169.2:9099
82df10ba-8143-400b-977a-8f5f416a4541    Service_openshif    tcp       172.30.26.159:2379
10.0.147.219:2379,10.0.163.212:2379,169.254.169.2:2379
                                        tcp       172.30.26.159:9979
10.0.147.219:9979,10.0.163.212:9979,169.254.169.2:9979
debe7f3a-39a8-490e-bc0a-ebbfafdffb16    Service_openshif    tcp       172.30.23.244:443
10.128.0.48:8443,10.129.0.27:8443,10.130.0.45:8443
8a749239-02d9-4dc2-8737-716528e0da7b    Service_openshif    tcp
172.30.124.255:8443    10.128.0.14:8443
880c7c78-c790-403d-a3cb-9f06592717a3    Service_openshif    tcp       172.30.0.10:53
10.130.0.20:5353
                                        tcp       172.30.0.10:9154       10.130.0.20:9154
d2f39078-6751-4311-a161-815bbaf7f9c7    Service_openshif    tcp       172.30.26.159:2379
169.254.169.2:2379,10.0.163.212:2379,10.0.242.240:2379
                                        tcp       172.30.26.159:9979
169.254.169.2:9979,10.0.163.212:9979,10.0.242.240:9979
30948278-602b-455c-934a-28e64c46de12    Service_openshif    tcp
172.30.157.35:9443       10.130.0.43:9443
2cc7e376-7c02-4a82-89e8-dfa1e23fb003    Service_openshif    tcp
172.30.159.212:17698    10.128.0.48:17698,10.129.0.27:17698,10.130.0.45:17698
e7d22d35-61c2-40c2-bc30-265cff8ed18d    Service_openshif    tcp       172.30.143.87:9001
10.0.145.205:9001,10.0.147.219:9001,10.0.163.212:9001,10.0.165.9:9001,10.0.209.170:9001,
169.254.169.2:9001
75164e75-e0c5-40fb-9636-bfdbf4223a02    Service_openshif    tcp       172.30.150.68:1936
10.129.4.8:1936,10.131.0.10:1936
                                        tcp       172.30.150.68:443
10.129.4.8:443,10.131.0.10:443
                                        tcp       172.30.150.68:80
10.129.4.8:80,10.131.0.10:80
7bc4ee74-dccf-47e9-9149-b011f09aff39    Service_openshif    tcp       172.30.164.74:443
10.0.147.219:6443,10.0.163.212:6443,169.254.169.2:6443
0db59e74-1cc6-470c-bf44-57c520e0aa8f    Service_openshif    tcp       10.0.163.212:31460
                                        tcp       10.0.163.212:32361
c300e134-018c-49af-9f84-9deb1d0715f8    Service_openshif    tcp       172.30.42.244:50051
10.130.0.47:50051
5e352773-429b-4881-afb3-a13b7ba8b081    Service_openshif    tcp       172.30.244.66:443
10.129.0.8:8443,10.130.0.8:8443
54b82d32-1939-4465-a87d-f26321442a7a    Service_openshif    tcp       172.30.12.9:8443
10.128.0.35:8443
```
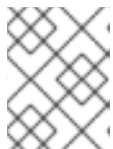
NOTE

From this truncated output you can see there are many OVN-Kubernetes load balancers. Load balancers in OVN-Kubernetes are representations of services.

27.2.4. Command line arguments for ovn-nbctl to examine northbound database contents

The following table describes the command line arguments that can be used with **ovn-nbctl** to examine the contents of the northbound database.

Table 27.2. Command line arguments to examine northbound database contents

Argument	Description
ovn-nbctl show	An overview of the northbound database contents.
ovn-nbctl show <switch_or_router>	Show the details associated with the specified switch or router.
ovn-nbctl lr-list	Show the logical routers.
ovn-nbctl lrp-list <router>	Using the router information from **ovn-nbctl lr-list** to show the router ports.
ovn-nbctl lr-nat-list <router>	Show network address translation details for the specified router.
ovn-nbctl ls-list	Show the logical switches
ovn-nbctl lsp-list <switch>	Using the switch information from **ovn-nbctl ls-list** to show the switch port.
ovn-nbctl lsp-get-type <port>	Get the type for the logical port.
ovn-nbctl lb-list	Show the load balancers.

27.2.5. Listing the OVN-Kubernetes southbound database contents

Logic flow rules are stored in the southbound database that is a representation of your infrastructure. The up to date information is present on the OVN Raft leader and this procedure describes how to find the Raft leader and query it to list the OVN southbound database contents.

Prerequisites

- Access to the cluster as a user with the **cluster-admin** role.

- The OpenShift CLI (**oc**) installed.

Procedure

1. Find the OVN Raft leader for the southbound database.

   > **NOTE**
   >
   > The Raft leader stores the most up to date information.

   a. List the pods by running the following command:

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

   Example output

```
NAME                 READY  STATUS   RESTARTS       AGE
ovnkube-master-7j97q  6/6    Running  2 (134m ago)   135m
ovnkube-master-gt4ms  6/6    Running  1 (126m ago)   133m
ovnkube-master-mk6p6  6/6    Running  0              134m
ovnkube-node-8qvtr    5/5    Running  0              135m
ovnkube-node-bqztb    5/5    Running  0              117m
ovnkube-node-fqdc9    5/5    Running  0              135m
ovnkube-node-tlfwv    5/5    Running  0              135m
ovnkube-node-wlwkn    5/5    Running  0              128m
```

b. Choose one of the master pods at random and run the following command to find the OVN southbound Raft leader:

```
$ oc exec -n openshift-ovn-kubernetes ovnkube-master-7j97q \
-- /usr/bin/ovn-appctl -t /var/run/ovn/ovnsb_db.ctl \
--timeout=3 cluster/status OVN_Southbound
```

Example output

```
Defaulted container "northd" out of: northd, nbdb, kube-rbac-proxy, sbdb, ovnkube-
master, ovn-dbchecker
1930
Name: OVN_Southbound
Cluster ID: f772 (f77273c0-7986-42dd-bd3c-a9f18e25701f)
Server ID: 1930 (1930f4b7-314b-406f-9dcb-b81fe2729ae1)
Address: ssl:10.0.147.219:9644
Status: cluster member
Role: follower ❶
Term: 3
Leader: 7081 ❷
Vote: unknown

Election timer: 16000
Log: [2, 2423]
Entries not yet committed: 0
Entries not yet applied: 0
Connections: ->0000 ->7145 <-7081 <-7145
Disconnections: 0
Servers:
    7081 (7081 at ssl:10.0.163.212:9644) last msg 59 ms ago ❸
    1930 (1930 at ssl:10.0.147.219:9644) (self)
    7145 (7145 at ssl:10.0.242.240:9644) last msg 7871735 ms ago
```

❶ This pod is identified as a follower

❷ The leader is identified as **7081**

❸ The **7081** is on IP address **10.0.163.212**

c. Find the **ovnkube-master** pod running on IP Address **10.0.163.212** using the following command:

```
$ oc get po -o wide -n openshift-ovn-kubernetes | grep 10.0.163.212 | grep -v ovnkube-
node
```

Example output

```
ovnkube-master-mk6p6   6/6     Running   0           136m   10.0.163.212   ip-10-0-163-
212.ec2.internal   <none>          <none>
```

The **ovnkube-master-mk6p6** pod runs on IP Address 10.0.163.212.

2. Run the following command to show all the information stored in the southbound database:

```
$ oc exec -n openshift-ovn-kubernetes -it ovnkube-master-mk6p6 \
-c northd -- ovn-sbctl show
```

Example output

```
Chassis "8ca57b28-9834-45f0-99b0-96486c22e1be"
    hostname: ip-10-0-156-16.ec2.internal
    Encap geneve
        ip: "10.0.156.16"
        options: {csum="true"}
    Port_Binding k8s-ip-10-0-156-16.ec2.internal
    Port_Binding etor-GR_ip-10-0-156-16.ec2.internal
    Port_Binding jtor-GR_ip-10-0-156-16.ec2.internal
    Port_Binding openshift-ingress-canary_ingress-canary-hsblx
    Port_Binding rtoj-GR_ip-10-0-156-16.ec2.internal
    Port_Binding openshift-monitoring_prometheus-adapter-658fc5967-9l46x
    Port_Binding rtoe-GR_ip-10-0-156-16.ec2.internal
    Port_Binding openshift-multus_network-metrics-daemon-77nvz
    Port_Binding openshift-ingress_router-default-64fd8c67c7-df598
    Port_Binding openshift-dns_dns-default-ttpcq
    Port_Binding openshift-monitoring_alertmanager-main-0
    Port_Binding openshift-e2e-loki_loki-promtail-g2pbh
    Port_Binding openshift-network-diagnostics_network-check-target-m6tn4
    Port_Binding openshift-monitoring_thanos-querier-75b5cf8dcb-qf8qj
    Port_Binding cr-rtos-ip-10-0-156-16.ec2.internal
    Port_Binding openshift-image-registry_image-registry-7b7bc44566-mp9b8
```

This detailed output shows the chassis and the ports that are attached to the chassis which in this case are all of the router ports and anything that runs like host networking. Any pods communicate out to the wider network using source network address translation (SNAT). Their IP address is translated into the IP address of the node that the pod is running on and then sent out into the network.

In addition to the chassis information the southbound database has all the logic flows and those logic flows are then sent to the **ovn-controller** running on each of the nodes. The **ovn-controller** translates the logic flows into open flow rules and ultimately programs **OpenvSwitch** so that your pods can then follow open flow rules and make it out of the network.

Run the following command to display the options available with the command **ovn-sbctl**:

```
$ oc exec -n openshift-ovn-kubernetes -it ovnkube-master-mk6p6 \
-c northd -- ovn-sbctl --help
```

–

27.2.6. Command line arguments for ovn-sbctl to examine southbound database contents

The following table describes the command line arguments that can be used with **ovn-sbctl** to examine the contents of the southbound database.

Table 27.3. Command line arguments to examine southbound database contents

Argument	Description
ovn-sbctl show	Overview of the southbound database contents.
ovn-sbctl list Port_Binding <port>	List the contents of southbound database for a the specified port .
ovn-sbctl dump-flows	List the logical flows.

27.2.7. OVN-Kubernetes logical architecture

OVN is a network virtualization solution. It creates logical switches and routers. These switches and routers are interconnected to create any network topologies. When you run **ovnkube-trace** with the log level set to 2 or 5 the OVN-Kubernetes logical components are exposed. The following diagram shows how the routers and switches are connected in OpenShift Container Platform.

Figure 27.2. OVN-Kubernetes router and switch components
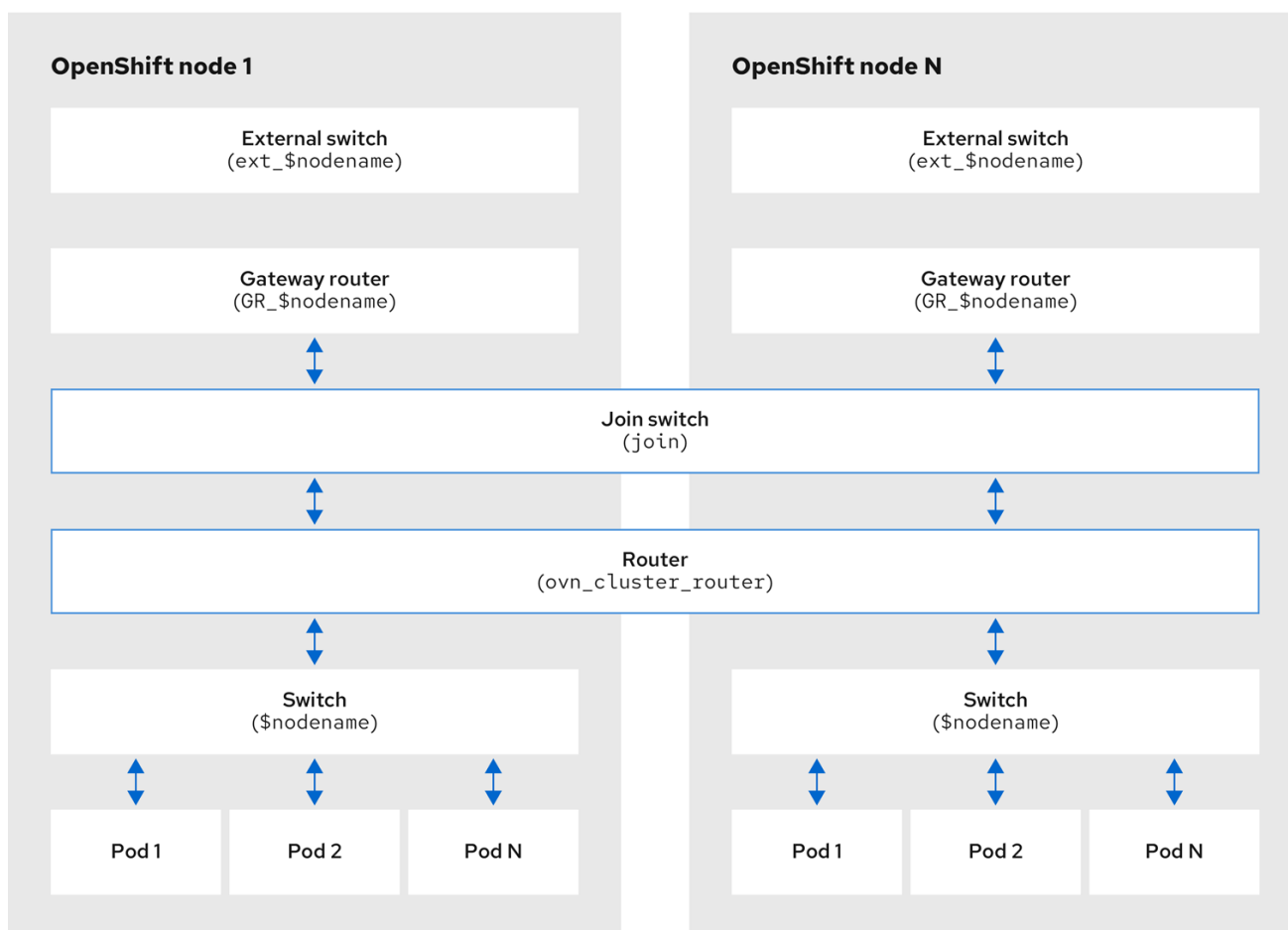
The key components involved in packet processing are:

Gateway routers

Gateway routers sometimes called L3 gateway routers, are typically used between the distributed routers and the physical network. Gateway routers including their logical patch ports are bound to a physical location (not distributed), or chassis. The patch ports on this router are known as l3gateway ports in the ovn-southbound database (**ovn-sbdb**).

Distributed logical routers

Distributed logical routers and the logical switches behind them, to which virtual machines and containers attach, effectively reside on each hypervisor.

Join local switch

Join local switches are used to connect the distributed router and gateway routers. It reduces the number of IP addresses needed on the distributed router.

Logical switches with patch ports

Logical switches with patch ports are used to virtualize the network stack. They connect remote logical ports through tunnels.

Logical switches with localnet ports

Logical switches with localnet ports are used to connect OVN to the physical network. They connect remote logical ports by bridging the packets to directly connected physical L2 segments using localnet ports.

Patch ports

Patch ports represent connectivity between logical switches and logical routers and between peer logical routers. A single connection has a pair of patch ports at each such point of connectivity, one on each side.

l3gateway ports

l3gateway ports are the port binding entries in the **ovn-sbdb** for logical patch ports used in the gateway routers. They are called l3gateway ports rather than patch ports just to portray the fact that these ports are bound to a chassis just like the gateway router itself.

localnet ports

localnet ports are present on the bridged logical switches that allows a connection to a locally accessible network from each **ovn-controller** instance. This helps model the direct connectivity to the physical network from the logical switches. A logical switch can only have a single localnet port attached to it.

27.2.7.1. Installing network-tools on local host

Install **network-tools** on your local host to make a collection of tools available for debugging OpenShift Container Platform cluster network issues.

Procedure

1. Clone the **network-tools** repository onto your workstation with the following command:

   ```
   $ git clone git@github.com:openshift/network-tools.git
   ```

2. Change into the directory for the repository you just cloned:

   ```
   $ cd network-tools
   ```

3. Optional: List all available commands:

   ```
   $ ./debug-scripts/network-tools -h
   ```

27.2.7.2. Running network-tools

Get information about the logical switches and routers by running **network-tools**.

Prerequisites

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

- You are logged in to the cluster as a user with **cluster-admin** privileges.

- You have installed **network-tools** on local host.

Procedure

1. List the routers by running the following command:

   ```
   $ ./debug-scripts/network-tools ovn-db-run-command ovn-nbctl lr-list
   ```

   Example output

```
Leader pod is ovnkube-master-vslqm
5351ddd1-f181-4e77-afc6-b48b0a9df953 (GR_helix13.lab.eng.tlv2.redhat.com)
ccf9349e-1948-4df8-954e-39fb0c2d4d06 (GR_helix14.lab.eng.tlv2.redhat.com)
e426b918-75a8-4220-9e76-20b7758f92b7 (GR_hlxcl7-master-
0.hlxcl7.lab.eng.tlv2.redhat.com)
dded77c8-0cc3-4b99-8420-56cd2ae6a840 (GR_hlxcl7-master-
1.hlxcl7.lab.eng.tlv2.redhat.com)
4f6747e6-e7ba-4e0c-8dcd-94c8efa51798 (GR_hlxcl7-master-
2.hlxcl7.lab.eng.tlv2.redhat.com)
52232654-336e-4952-98b9-0b8601e370b4 (ovn_cluster_router)
```

2. List the localnet ports by running the following command:

```
$ ./debug-scripts/network-tools ovn-db-run-command \
ovn-sbctl find Port_Binding type=localnet
```

Example output

```
Leader pod is ovnkube-master-vslqm
_uuid               : 3de79191-cca8-4c28-be5a-a228f0f9ebfc
additional_chassis  : []
additional_encap    : []
chassis             : []
datapath            : 3f1a4928-7ff5-471f-9092-fe5f5c67d15c
encap               : []
external_ids        : {}
gateway_chassis     : []
ha_chassis_group    : []
logical_port        : br-ex_helix13.lab.eng.tlv2.redhat.com
mac                 : [unknown]
nat_addresses       : []
options             : {network_name=physnet}
parent_port         : []
port_security       : []
requested_additional_chassis: []
requested_chassis   : []
tag                 : []
tunnel_key          : 2
type                : localnet
up                  : false
virtual_parent      : []

_uuid               : dbe21daf-9594-4849-b8f0-5efbfa09a455
additional_chassis  : []
additional_encap    : []
chassis             : []
datapath            : db2a6067-fe7c-4d11-95a7-ff2321329e11
encap               : []
external_ids        : {}
gateway_chassis     : []
ha_chassis_group    : []
logical_port        : br-ex_hlxcl7-master-2.hlxcl7.lab.eng.tlv2.redhat.com
mac                 : [unknown]
nat_addresses       : []
options             : {network_name=physnet}
```

```
parent_port        : []
port_security      : []
requested_additional_chassis: []
requested_chassis  : []
tag                : []
tunnel_key         : 2
type               : localnet
up                 : false
virtual_parent     : []

[...]
```

3. List the **l3gateway** ports by running the following command:

```
$ ./debug-scripts/network-tools ovn-db-run-command \
ovn-sbctl find Port_Binding type=l3gateway
```

Example output

```
Leader pod is ovnkube-master-vslqm
_uuid              : 9314dc80-39e1-4af7-9cc0-ae8a9708ed59
additional_chassis : []
additional_encap   : []
chassis            : 336a923d-99e8-4e71-89a6-12564fde5760
datapath           : db2a6067-fe7c-4d11-95a7-ff2321329e11
encap              : []
external_ids       : {}
gateway_chassis    : []
ha_chassis_group   : []
logical_port       : etor-GR_hlxcl7-master-2.hlxcl7.lab.eng.tlv2.redhat.com
mac                : ["52:54:00:3e:95:d3"]
nat_addresses      : ["52:54:00:3e:95:d3 10.46.56.77"]
options            : {l3gateway-chassis="7eb1f1c3-87c2-4f68-8e89-60f5ca810971", peer=rtoe-
GR_hlxcl7-master-2.hlxcl7.lab.eng.tlv2.redhat.com}
parent_port        : []
port_security      : []
requested_additional_chassis: []
requested_chassis  : []
tag                : []
tunnel_key         : 1
type               : l3gateway
up                 : true
virtual_parent     : []

_uuid              : ad7eb303-b411-4e9f-8d36-d07f1f268e27
additional_chassis : []
additional_encap   : []
chassis            : f41453b8-29c5-4f39-b86b-e82cf344bce4
datapath           : 082e7a60-d9c7-464b-b6ec-117d3426645a
encap              : []
external_ids       : {}
gateway_chassis    : []
ha_chassis_group   : []
logical_port       : etor-GR_helix14.lab.eng.tlv2.redhat.com
mac                : ["34:48:ed:f3:e2:2c"]
```

```
nat_addresses      : ["34:48:ed:f3:e2:2c 10.46.56.14"]
options            : {l3gateway-chassis="2e8abe3a-cb94-4593-9037-f5f9596325e2", peer=rtoe-
GR_helix14.lab.eng.tlv2.redhat.com}
parent_port        : []
port_security      : []
requested_additional_chassis: []
requested_chassis  : []
tag                : []
tunnel_key         : 1
type               : l3gateway
up                 : true
virtual_parent     : []

[...]
```

4. List the patch ports by running the following command:

```
$ ./debug-scripts/network-tools ovn-db-run-command \
ovn-sbctl find Port_Binding type=patch
```

Example output

```
Leader pod is ovnkube-master-vslqm
_uuid             : c48b1380-ff26-4965-a644-6bd5b5946c61
additional_chassis  : []
additional_encap    : []
chassis           : []
datapath          : 72734d65-fae1-4bd9-a1ee-1bf4e085a060
encap             : []
external_ids      : {}
gateway_chassis     : []
ha_chassis_group    : []
logical_port      : jtor-ovn_cluster_router
mac               : [router]
nat_addresses     : []
options           : {peer=rtoj-ovn_cluster_router}
parent_port       : []
port_security     : []
requested_additional_chassis: []
requested_chassis   : []
tag               : []
tunnel_key        : 4
type              : patch
up                : false
virtual_parent    : []

_uuid             : 5df51302-f3cd-415b-a059-ac24389938f7
additional_chassis  : []
additional_encap    : []
chassis           : []
datapath          : 0551c90f-e891-4909-8e9e-acc7909e06d0
encap             : []
external_ids      : {}
gateway_chassis     : []
ha_chassis_group    : []
```

```
logical_port      : rtos-hlxcl7-master-1.hlxcl7.lab.eng.tlv2.redhat.com
mac               : ["0a:58:0a:82:00:01 10.130.0.1/23"]
nat_addresses     : []
options           : {chassis-redirect-port=cr-rtos-hlxcl7-master-1.hlxcl7.lab.eng.tlv2.redhat.com,
peer=stor-hlxcl7-master-1.hlxcl7.lab.eng.tlv2.redhat.com}
parent_port       : []
port_security     : []
requested_additional_chassis: []
requested_chassis   : []
tag               : []
tunnel_key        : 4
type              : patch
up                : false
virtual_parent    : []

[...]
```

27.2.8. Additional resources

- How to list OVN database contents with ovn-kubernetes in Red Hat OpenShift Container Platform 4.x?

- Tracing Openflow with ovnkube-trace

- OVN architecture

- Raft (algorithm)

- ovn-nbctl linux manual page

- ovn-sbctl linux manual page

27.3. TROUBLESHOOTING OVN-KUBERNETES

OVN-Kubernetes has many sources of built-in health checks and logs.

27.3.1. Monitoring OVN-Kubernetes health by using readiness probes

The **ovnkube-master** and **ovnkube-node** pods have containers configured with readiness probes.

Prerequisites

- Access to the OpenShift CLI (**oc**).

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

- You have installed **jq**.

Procedure

1. Review the details of the **ovnkube-master** readiness probe by running the following command:

   ```
   $ oc get pods -n openshift-ovn-kubernetes -l app=ovnkube-master \
   -o json | jq '.items[0].spec.containers[] | .name,.readinessProbe'
   ```

The readiness probe for the northbound and southbound database containers in the **ovnkube-master** pod checks for the health of the Raft cluster hosting the databases.

2. Review the details of the **ovnkube-node** readiness probe by running the following command:

```
$ oc get pods -n openshift-ovn-kubernetes -l app=ovnkube-master \
-o json | jq '.items[0].spec.containers[] | .name,.readinessProbe'
```

The **ovnkube-node** container in the **ovnkube-node** pod has a readiness probe to verify the presence of the ovn-kubernetes CNI configuration file, the absence of which would indicate that the pod is not running or is not ready to accept requests to configure pods.

3. Show all events including the probe failures, for the namespace by using the following command:

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

4. Show the events for just this pod:

```
$ oc describe pod ovnkube-master-tp2z8 -n openshift-ovn-kubernetes
```

5. Show the messages and statuses from the cluster network operator:

```
$ oc get co/network -o json | jq '.status.conditions[]'
```

6. Show the **ready** status of each container in **ovnkube-master** pods by running the following script:

```
$ for p in $(oc get pods --selector app=ovnkube-master -n openshift-ovn-kubernetes \
-o jsonpath='{range.items[*]}{" "}{.metadata.name}'); do echo === $p ===;  \
oc get pods -n openshift-ovn-kubernetes $p -o json | jq '.status.containerStatuses[] | .name,
.ready'; \
done
```

> **NOTE**
>
> The expectation is all container statuses are reporting as **true**. Failure of a readiness probe sets the status to **false**.

Additional resources

- [Monitoring application health by using health checks](#)

27.3.2. Viewing OVN-Kubernetes alerts in the console

The Alerting UI provides detailed information about alerts and their governing alerting rules and silences.

Prerequisites

- You have access to the cluster as a developer or as a user with view permissions for the project that you are viewing metrics for.

Procedure (UI)

1. In the **Administrator** perspective, select **Observe → Alerting**. The three main pages in the Alerting UI in this perspective are the **Alerts**, **Silences**, and **Alerting Rules** pages.

2. View the rules for OVN-Kubernetes alerts by selecting **Observe → Alerting → Alerting Rules**.

27.3.3. Viewing OVN-Kubernetes alerts in the CLI

You can get information about alerts and their governing alerting rules and silences from the command line.

Prerequisites

- Access to the cluster as a user with the **cluster-admin** role.

- The OpenShift CLI (**oc**) installed.

- You have installed **jq**.

Procedure

1. View active or firing alerts by running the following commands.

    a. Set the alert manager route environment variable by running the following command:

    ```
    $ ALERT_MANAGER=$(oc get route alertmanager-main -n openshift-monitoring \
    -o jsonpath='{@.spec.host}')
    ```

    b. Issue a **curl** request to the alert manager route API with the correct authorization details requesting specific fields by running the following command:

    ```
    $ curl -s -k -H "Authorization: Bearer \
    $(oc create token prometheus-k8s -n openshift-monitoring)" \
    https://$ALERT_MANAGER/api/v1/alerts \
    | jq '.data[] | "\(.labels.severity) \(.labels.alertname) \(.labels.pod) \(.labels.container) \
    (.labels.endpoint) \(.labels.instance)"'
    ```

2. View alerting rules by running the following command:

    ```
    $ oc -n openshift-monitoring exec -c prometheus prometheus-k8s-0 -- curl -s
    'http://localhost:9090/api/v1/rules' | jq '.data.groups[].rules[] | select(((.name|contains("ovn"))
    or (.name|contains("OVN")) or (.name|contains("Ovn")) or (.name|contains("North")) or
    (.name|contains("South"))) and .type=="alerting")'
    ```

27.3.4. Viewing the OVN-Kubernetes logs using the CLI

You can view the logs for each of the pods in the **ovnkube-master** and **ovnkube-node** pods using the OpenShift CLI (**oc**).

Prerequisites

- Access to the cluster as a user with the **cluster-admin** role.

- Access to the OpenShift CLI (**oc**).

- You have installed **jq**.

Procedure

1. View the log for a specific pod:

   ```
   $ oc logs -f <pod_name> -c <container_name> -n <namespace>
   ```

   where:

   -f

   Optional: Specifies that the output follows what is being written into the logs.

   <pod_name>

   Specifies the name of the pod.

   <container_name>

   Optional: Specifies the name of a container. When a pod has more than one container, you must specify the container name.

   <namespace>

   Specify the namespace the pod is running in.

   For example:

   ```
   $ oc logs ovnkube-master-7h4q7 -n openshift-ovn-kubernetes
   ```

   ```
   $ oc logs -f ovnkube-master-7h4q7 -n openshift-ovn-kubernetes -c ovn-dbchecker
   ```

   The contents of log files are printed out.

2. Examine the most recent entries in all the containers in the **ovnkube-master** pods:

   ```
   $ for p in $(oc get pods --selector app=ovnkube-master -n openshift-ovn-kubernetes \
   -o jsonpath='{range.items[*]}{" "}{.metadata.name}'); \
   do echo === $p ===; for container in $(oc get pods -n openshift-ovn-kubernetes $p \
   -o json | jq -r '.status.containerStatuses[] | .name');do echo ---$container---; \
   oc logs -c $container $p -n openshift-ovn-kubernetes --tail=5; done; done
   ```

3. View the last 5 lines of every log in every container in an **ovnkube-master** pod using the following command:

   ```
   $ oc logs -l app=ovnkube-master -n openshift-ovn-kubernetes --all-containers --tail 5
   ```

27.3.5. Viewing the OVN-Kubernetes logs using the web console

You can view the logs for each of the pods in the **ovnkube-master** and **ovnkube-node** pods in the web console.

Prerequisites

- Access to the OpenShift CLI (**oc**).

Procedure

1. In the OpenShift Container Platform console, navigate to **Workloads → Pods** or navigate to the pod through the resource you want to investigate.

2. Select the **openshift-ovn-kubernetes** project from the drop-down menu.

3. Click the name of the pod you want to investigate.

4. Click **Logs**. By default for the **ovnkube-master** the logs associated with the **northd** container are displayed.

5. Use the down-down menu to select logs for each container in turn.

27.3.5.1. Changing the OVN-Kubernetes log levels

The default log level for OVN-Kubernetes is 2. To debug OVN-Kubernetes set the log level to 5. Follow this procedure to increase the log level of the OVN-Kubernetes to help you debug an issue.

Prerequisites

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

- You have access to the OpenShift Container Platform web console.

Procedure

1. Run the following command to get detailed information for all pods in the OVN-Kubernetes project:

   ```
   $ oc get po -o wide -n openshift-ovn-kubernetes
   ```

   Example output

   ```
   NAME                READY STATUS   RESTARTS    AGE  IP          NODE
   NOMINATED NODE   READINESS GATES
   ovnkube-master-84nc9  6/6    Running  0          50m  10.0.134.156  ip-10-0-134-
   156.ec2.internal  <none>        <none>
   ovnkube-master-gmlqv  6/6    Running  0          50m  10.0.209.180  ip-10-0-209-
   180.ec2.internal  <none>        <none>
   ovnkube-master-nhts2  6/6    Running  1 (48m ago)  50m  10.0.147.31   ip-10-0-147-
   31.ec2.internal   <none>        <none>
   ovnkube-node-2cbh8    5/5    Running  0          43m  10.0.217.114  ip-10-0-217-
   114.ec2.internal  <none>        <none>
   ovnkube-node-6fvzl    5/5    Running  0          50m  10.0.147.31   ip-10-0-147-
   31.ec2.internal   <none>        <none>
   ovnkube-node-f4lzz    5/5    Running  0          24m  10.0.146.76   ip-10-0-146-
   76.ec2.internal   <none>        <none>
   ovnkube-node-jf67d    5/5    Running  0          50m  10.0.209.180  ip-10-0-209-
   180.ec2.internal  <none>        <none>
   ovnkube-node-np9mf    5/5    Running  0          40m  10.0.165.191  ip-10-0-165-
   191.ec2.internal  <none>        <none>
   ovnkube-node-qjldg    5/5    Running  0          50m  10.0.134.156  ip-10-0-134-
   156.ec2.internal  <none>        <none>
   ```

2. Create a **ConfigMap** file similar to the following example and use a filename such as **env-overrides.yaml**:

Example **ConfigMap** file

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: env-overrides
  namespace: openshift-ovn-kubernetes
data:
  ip-10-0-217-114.ec2.internal: | 1
    # This sets the log level for the ovn-kubernetes node process:
    OVN_KUBE_LOG_LEVEL=5
    # You might also/instead want to enable debug logging for ovn-controller:
    OVN_LOG_LEVEL=dbg
  ip-10-0-209-180.ec2.internal: |
    # This sets the log level for the ovn-kubernetes node process:
    OVN_KUBE_LOG_LEVEL=5
    # You might also/instead want to enable debug logging for ovn-controller:
    OVN_LOG_LEVEL=dbg
  _master: | 2
    # This sets the log level for the ovn-kubernetes master process as well as the ovn-
dbchecker:
    OVN_KUBE_LOG_LEVEL=5
    # You might also/instead want to enable debug logging for northd, nbdb and sbdb on all
masters:
    OVN_LOG_LEVEL=dbg
```

[1] Specify the name of the node you want to set the debug log level on.

[2] Specify **_master** to set the log levels of **ovnkube-master** components.

3. Apply the **ConfigMap** file by using the following command:

```
$ oc apply -n openshift-ovn-kubernetes -f env-overrides.yaml
```

Example output

```
configmap/env-overrides.yaml created
```

4. Restart the **ovnkube** pods to apply the new log level by using the following commands:

```
$ oc delete pod -n openshift-ovn-kubernetes \
--field-selector spec.nodeName=ip-10-0-217-114.ec2.internal -l app=ovnkube-node
```

```
$ oc delete pod -n openshift-ovn-kubernetes \
--field-selector spec.nodeName=ip-10-0-209-180.ec2.internal -l app=ovnkube-node
```

```
$ oc delete pod -n openshift-ovn-kubernetes -l app=ovnkube-master
```

27.3.6. Checking the OVN-Kubernetes pod network connectivity

The connectivity check controller, in OpenShift Container Platform 4.10 and later, orchestrates connection verification checks in your cluster. These include Kubernetes API, OpenShift API and individual nodes. The results for the connection tests are stored in **PodNetworkConnectivity** objects in the **openshift-network-diagnostics** namespace. Connection tests are performed every minute in parallel.

Prerequisites

- Access to the OpenShift CLI (**oc**).

- Access to the cluster as a user with the **cluster-admin** role.

- You have installed **jq**.

Procedure

1. To list the current **PodNetworkConnectivityCheck** objects, enter the following command:

   ```
   $ oc get podnetworkconnectivitychecks -n openshift-network-diagnostics
   ```

2. View the most recent success for each connection object by using the following command:

   ```
   $ oc get podnetworkconnectivitychecks -n openshift-network-diagnostics \
   -o json | jq '.items[]| .spec.targetEndpoint,.status.successes[0]'
   ```

3. View the most recent failures for each connection object by using the following command:

   ```
   $ oc get podnetworkconnectivitychecks -n openshift-network-diagnostics \
   -o json | jq '.items[]| .spec.targetEndpoint,.status.failures[0]'
   ```

4. View the most recent outages for each connection object by using the following command:

   ```
   $ oc get podnetworkconnectivitychecks -n openshift-network-diagnostics \
   -o json | jq '.items[]| .spec.targetEndpoint,.status.outages[0]'
   ```

   The connectivity check controller also logs metrics from these checks into Prometheus.

5. View all the metrics by running the following command:

   ```
   $ oc exec prometheus-k8s-0 -n openshift-monitoring -- \
   promtool query instant  http://localhost:9090 \
   '{component="openshift-network-diagnostics"}'
   ```

6. View the latency between the source pod and the openshift api service for the last 5 minutes:

   ```
   $ oc exec prometheus-k8s-0 -n openshift-monitoring -- \
   promtool query instant  http://localhost:9090 \
   '{component="openshift-network-diagnostics"}'
   ```

27.3.7. Additional resources

- How do I change the ovn-kubernetes loglevel in OpenShift 4?

- Implementation of connection health checks

- Verifying network connectivity for an endpoint

27.4. TRACING OPENFLOW WITH OVNKUBE-TRACE

OVN and OVS traffic flows can be simulated in a single utility called **ovnkube-trace**. The **ovnkube-trace** utility runs **ovn-trace**, **ovs-appctl ofproto/trace** and **ovn-detrace** and correlates that information in a single output.

You can execute the **ovnkube-trace** binary from a dedicated container. For releases after OpenShift Container Platform 4.7, you can also copy the binary to a local host and execute it from that host.

> **NOTE**
>
> The binaries in the Quay images do not currently work for Dual IP stack or IPv6 only environments. For those environments, you must build from source.

27.4.1. Installing the ovnkube-trace on local host

The **ovnkube-trace** tool traces packet simulations for arbitrary UDP or TCP traffic between points in an OVN-Kubernetes driven OpenShift Container Platform cluster. Copy the **ovnkube-trace** binary to your local host making it available to run against the cluster.

Prerequisites

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

- You are logged in to the cluster with a user with **cluster-admin** privileges.

Procedure

1. Create a pod variable by using the following command:

   ```
   $ POD=$(oc get pods -n openshift-ovn-kubernetes -l app=ovnkube-master -o name | head -1 | awk -F '/' '{print $NF}')
   ```

2. Run the following command on your local host to copy the binary from the **ovnkube-master** pods:

   ```
   $ oc cp -n openshift-ovn-kubernetes $POD:/usr/bin/ovnkube-trace ovnkube-trace
   ```

3. Make **ovnkube-trace** executable by running the following command:

   ```
   $ chmod +x ovnkube-trace
   ```

4. Display the options available with **ovnkube-trace** by running the following command:

   ```
   $ ./ovnkube-trace -help
   ```

   ### Expected output

```
I0111 15:05:27.973305  204872 ovs.go:90] Maximum command line arguments set to:
191102
Usage of ./ovnkube-trace:
 -dst string
    dest: destination pod name
 -dst-ip string
    destination IP address (meant for tests to external targets)
 -dst-namespace string
    k8s namespace of dest pod (default "default")
 -dst-port string
    dst-port: destination port (default "80")
 -kubeconfig string
    absolute path to the kubeconfig file
 -loglevel string
    loglevel: klog level (default "0")
 -ovn-config-namespace string
    namespace used by ovn-config itself
 -service string
    service: destination service name
 -skip-detrace
    skip ovn-detrace command
 -src string
    src: source pod name
 -src-namespace string
    k8s namespace of source pod (default "default")
 -tcp
    use tcp transport protocol
 -udp
    use udp transport protocol
```

The command-line arguments supported are familiar Kubernetes constructs, such as namespaces, pods, services so you do not need to find the MAC address, the IP address of the destination nodes, or the ICMP type.

The log levels are:

- 0 (minimal output)

- 2 (more verbose output showing results of trace commands)

- 5 (debug output)

27.4.2. Running ovnkube-trace

Run **ovn-trace** to simulate packet forwarding within an OVN logical network.

Prerequisites

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

- You are logged in to the cluster with a user with **cluster-admin** privileges.

- You have installed **ovnkube-trace** on local host

Example: Testing that DNS resolution works from a deployed pod

This example illustrates how to test the DNS resolution from a deployed pod to the core DNS pod that runs in the cluster.

Procedure

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. List the pods running in the **openshift-dns** namespace:

   ```
   oc get pods -n openshift-dns
   ```

   Example output

   ```
   NAME                READY   STATUS    RESTARTS   AGE
   dns-default-467qw    2/2     Running   0          49m
   dns-default-6prvx    2/2     Running   0          53m
   dns-default-fkqr8    2/2     Running   0          53m
   dns-default-qv2rg    2/2     Running   0          49m
   dns-default-s29vr    2/2     Running   0          49m
   dns-default-vdsbn    2/2     Running   0          53m
   node-resolver-6thtt  1/1     Running   0          53m
   node-resolver-7ksdn  1/1     Running   0          49m
   node-resolver-8sthh  1/1     Running   0          53m
   node-resolver-c5ksw  1/1     Running   0          50m
   node-resolver-gbvdp  1/1     Running   0          53m
   node-resolver-sxhkd  1/1     Running   0          50m
   ```

3. Run the following **ovn-kube-trace** command to verify DNS resolution is working:

   ```
   $ ./ovnkube-trace \
     -src-namespace default \      1
     -src web \      2
     -dst-namespace openshift-dns \      3
     -dst dns-default-467qw \      4
     -udp -dst-port 53 \      5
     -loglevel 0      6
   ```

   1 Namespace of the source pod

   2 Source pod name

   3 Namespace of destination pod

   4 Destination pod name

   5 Use the **udp** transport protocol. Port 53 is the port the DNS service uses.

   6 Set the log level to 1 (0 is minimal and 5 is debug)

   Expected output

```
I0116 10:19:35.601303   17900 ovs.go:90] Maximum command line arguments set to:
191102
ovn-trace source pod to destination pod indicates success from web to dns-default-467qw
ovn-trace destination pod to source pod indicates success from dns-default-467qw to web
ovs-appctl ofproto/trace source pod to destination pod indicates success from web to dns-
default-467qw
ovs-appctl ofproto/trace destination pod to source pod indicates success from dns-default-
467qw to web
ovn-detrace source pod to destination pod indicates success from web to dns-default-467qw
ovn-detrace destination pod to source pod indicates success from dns-default-467qw to web
```

The ouput indicates success from the deployed pod to the DNS port and also indicates that it is successful going back in the other direction. So you know bi-directional traffic is supported on UDP port 53 if my web pod wants to do dns resolution from core DNS.

If for example that did not work and you wanted to get the **ovn-trace**, the **ovs-appctl ofproto/trace** and **ovn-detrace**, and more debug type information increase the log level to 2 and run the command again as follows:

```
$ ./ovnkube-trace \
  -src-namespace default \
  -src web \
  -dst-namespace openshift-dns \
  -dst dns-default-467qw \
  -udp -dst-port 53 \
  -loglevel 2
```

The output from this increased log level is too much to list here. In a failure situation the output of this command shows which flow is dropping that traffic. For example an egress or ingress network policy may be configured on the cluster that does not allow that traffic.

Example: Verifying by using debug output a configured default deny

This example illustrates how to identify by using the debug output that an ingress default deny policy blocks traffic.

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
spec:
  podSelector: {}
  ingress: []
```

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. 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

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

   > $ oc create namespace prod

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

   > $ oc label namespace/prod purpose=production

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

   > $ oc run test-6459 --namespace=prod --rm -i -t --image=alpine -- sh

7. Open another terminal session.

8. In this new terminal session run **ovn-trace** to verify the failure in communication between the source pod **test-6459** running in namespace **prod** and destination pod running in the **default** namespace:

   > $ ./ovnkube-trace \
   > -src-namespace prod \
   > -src test-6459 \
   > -dst-namespace default \
   > -dst web \
   > -tcp -dst-port 80 \
   > -loglevel 0

   Expected output

   > I0116 14:20:47.380775   50822 ovs.go:90] Maximum command line arguments set to: 191102
   > ovn-trace source pod to destination pod indicates failure from test-6459 to web

9. Increase the log level to 2 to expose the reason for the failure by running the following command:

   > $ ./ovnkube-trace \
   > -src-namespace prod \
   > -src test-6459 \
   > -dst-namespace default \
   > -dst web \
   > -tcp -dst-port 80 \
   > -loglevel 2

   Expected output

   > ct_lb_mark /* default (use --ct to customize) */

```
-----------------------------------------------
 3. ls_out_acl_hint (northd.c:6092): !ct.new && ct.est && !ct.rpl && ct_mark.blocked == 0,
priority 4, uuid 32d45ad4
    reg0[8] = 1;
    reg0[10] = 1;
    next;
 4. ls_out_acl (northd.c:6435): reg0[10] == 1 && (outport ==
@a16982411286042166782_ingressDefaultDeny), priority 2000, uuid f730a887
    ct_commit { ct_mark.blocked = 1; };
```

1

1   Ingress traffic is blocked due to the default deny policy being in place

10. 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
      policyTypes:
      - Ingress
      ingress:
      - from:
        - namespaceSelector:
            matchLabels:
              purpose: production
    ```

11. Apply the policy by entering the following command:

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

12. Run **ovnkube-trace** to verify that traffic is now allowed by entering the following command:

    ```
    $ ./ovnkube-trace \
     -src-namespace prod \
     -src test-6459 \
     -dst-namespace default \
     -dst web \
     -tcp -dst-port 80 \
     -loglevel 0
    ```

    Expected output

    ```
    I0116 14:25:44.055207   51695 ovs.go:90] Maximum command line arguments set to:
    191102
    ovn-trace source pod to destination pod indicates success from test-6459 to web
    ovn-trace destination pod to source pod indicates success from web to test-6459
    ovs-appctl ofproto/trace source pod to destination pod indicates success from test-6459 to
    ```

```
web
ovs-appctl ofproto/trace destination pod to source pod indicates success from web to test-
6459
ovn-detrace source pod to destination pod indicates success from test-6459 to web
ovn-detrace destination pod to source pod indicates success from web to test-6459
```

13. In the open shell run the following command:

```
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>
```

27.4.3. Additional resources

- Tracing Openflow with ovnkube-trace utility

- ovnkube-trace

27.5. MIGRATING FROM THE OPENSHIFT SDN NETWORK PLUGIN

As a cluster administrator, you can migrate to the OVN-Kubernetes network plugin from the OpenShift SDN network plugin.

To learn more about OVN-Kubernetes, read About the OVN-Kubernetes network plugin .

27.5.1. Migration to the OVN-Kubernetes network plugin

Migrating to the OVN-Kubernetes network plugin is a manual process that includes some downtime during which your cluster is unreachable. Although a rollback procedure is provided, the migration is intended to be a one-way process.

A migration to the OVN-Kubernetes network plugin is supported on the following platforms:

- Bare metal hardware

- Amazon Web Services (AWS)

- Google Cloud Platform (GCP)

- IBM Cloud

- Microsoft Azure

- Red Hat OpenStack Platform (RHOSP)

- Red Hat Virtualization (RHV)

- VMware vSphere

> **IMPORTANT**
>
> Migrating to or from the OVN-Kubernetes network plugin is not supported for managed OpenShift cloud services such as Red Hat OpenShift Dedicated, Azure Red Hat OpenShift(ARO), and Red Hat OpenShift Service on AWS (ROSA).
>
> Migrating from OpenShift SDN network plugin to OVN-Kubernetes network plugin is not supported on Nutanix.

27.5.1.1. Considerations for migrating to the OVN-Kubernetes network plugin

If you have more than 150 nodes in your OpenShift Container Platform cluster, then open a support case for consultation on your migration to the OVN-Kubernetes network plugin.

The subnets assigned to nodes and the IP addresses assigned to individual pods are not preserved during the migration.

While the OVN-Kubernetes network plugin implements many of the capabilities present in the OpenShift SDN network plugin, the configuration is not the same.

- If your cluster uses any of the following OpenShift SDN network plugin capabilities, you must manually configure the same capability in the OVN-Kubernetes network plugin:

    - Namespace isolation

    - Egress router pods

- If your cluster or surrounding network uses any part of the **100.64.0.0/16** address range, you must choose another unused IP range by specifying the **v4InternalSubnet** spec under the **spec.defaultNetwork.ovnKubernetesConfig** object definition. OVN-Kubernetes uses the IP range **100.64.0.0/16** internally by default.

The following sections highlight the differences in configuration between the aforementioned capabilities in OVN-Kubernetes and OpenShift SDN network plugins.

Namespace isolation
OVN-Kubernetes supports only the network policy isolation mode.

> **IMPORTANT**
>
> If your cluster uses OpenShift SDN configured in either the multitenant or subnet isolation modes, you cannot migrate to the OVN-Kubernetes network plugin.

Egress IP addresses

OpenShift SDN supports two different Egress IP modes:

- In the *automatically assigned* approach, an egress IP address range is assigned to a node.

- In the *manually assigned* approach, a list of one or more egress IP addresses is assigned to a node.

The migration process supports migrating Egress IP configurations that use the automatically assigned mode.

The differences in configuring an egress IP address between OVN-Kubernetes and OpenShift SDN is described in the following table:

Table 27.4. Differences in egress IP address configuration

OVN-Kubernetes	OpenShift SDN
- Create an **EgressIPs** object - Add an annotation on a **Node** object	- Patch a **NetNamespace** object - Patch a **HostSubnet** object

For more information on using egress IP addresses in OVN-Kubernetes, see "Configuring an egress IP address".

Egress network policies

The difference in configuring an egress network policy, also known as an egress firewall, between OVN-Kubernetes and OpenShift SDN is described in the following table:

Table 27.5. Differences in egress network policy configuration

OVN-Kubernetes	OpenShift SDN
Create an **EgressFirewall** object in a namespace	Create an **EgressNetworkPolicy** object in a namespace

NOTE

Because the name of an **EgressFirewall** object can only be set to **default**, after the migration all migrated **EgressNetworkPolicy** objects are named **default**, regardless of what the name was under OpenShift SDN.

If you subsequently rollback to OpenShift SDN, all **EgressNetworkPolicy** objects are named **default** as the prior name is lost.

For more information on using an egress firewall in OVN-Kubernetes, see "Configuring an egress firewall for a project".

Egress router pods
OVN-Kubernetes supports egress router pods in redirect mode. OVN-Kubernetes does not support egress router pods in HTTP proxy mode or DNS proxy mode.

When you deploy an egress router with the Cluster Network Operator, you cannot specify a node selector to control which node is used to host the egress router pod.

Multicast
The difference between enabling multicast traffic on OVN-Kubernetes and OpenShift SDN is described in the following table:

Table 27.6. Differences in multicast configuration

OVN-Kubernetes	OpenShift SDN
• Add an annotation on a **Namespace** object	• Add an annotation on a **NetNamespace** object

For more information on using multicast in OVN-Kubernetes, see "Enabling multicast for a project".

Network policies
OVN-Kubernetes fully supports the Kubernetes **NetworkPolicy** API in the **networking.k8s.io/v1** API group. No changes are necessary in your network policies when migrating from OpenShift SDN.

27.5.1.2. How the migration process works

The following table summarizes the migration process by segmenting between the user-initiated steps in the process and the actions that the migration performs in response.

Table 27.7. Migrating to OVN-Kubernetes from OpenShift SDN

User-initiated steps	Migration activity

User-initiated steps	Migration activity
Set the **migration** field of the **Network.operator.openshift.io** custom resource (CR) named **cluster** to **OVNKubernetes**. Make sure the **migration** field is **null** before setting it to a value.	**Cluster Network Operator (CNO)** Updates the status of the **Network.config.openshift.io** CR named **cluster** accordingly. **Machine Config Operator (MCO)** Rolls out an update to the systemd configuration necessary for OVN-Kubernetes; the MCO updates a single machine per pool at a time by default, causing the total time the migration takes to increase with the size of the cluster.
Update the **networkType** field of the **Network.config.openshift.io** CR.	**CNO** Performs the following actions: • Destroys the OpenShift SDN control plane pods. • Deploys the OVN-Kubernetes control plane pods. • Updates the Multus objects to reflect the new network plugin.
Reboot each node in the cluster.	**Cluster** As nodes reboot, the cluster assigns IP addresses to pods on the OVN-Kubernetes cluster network.

If a rollback to OpenShift SDN is required, the following table describes the process.

Table 27.8. Performing a rollback to OpenShift SDN

User-initiated steps	Migration activity
Suspend the MCO to ensure that it does not interrupt the migration.	The MCO stops.
Set the **migration** field of the **Network.operator.openshift.io** custom resource (CR) named **cluster** to **OpenShiftSDN**. Make sure the **migration** field is **null** before setting it to a value.	**CNO** Updates the status of the **Network.config.openshift.io** CR named **cluster** accordingly.

User-initiated steps	Migration activity
Update the **networkType** field.	**CNO** Performs the following actions: • Destroys the OVN-Kubernetes control plane pods. • Deploys the OpenShift SDN control plane pods. • Updates the Multus objects to reflect the new network plugin.
Reboot each node in the cluster.	**Cluster** As nodes reboot, the cluster assigns IP addresses to pods on the OpenShift-SDN network.
Enable the MCO after all nodes in the cluster reboot.	**MCO** Rolls out an update to the systemd configuration necessary for OpenShift SDN; the MCO updates a single machine per pool at a time by default, so the total time the migration takes increases with the size of the cluster.

27.5.2. Migrating to the OVN-Kubernetes network plugin

As a cluster administrator, you can change the network plugin for your cluster to OVN-Kubernetes. During the migration, you must reboot every node in your cluster.

IMPORTANT

While performing the migration, your cluster is unavailable and workloads might be interrupted. Perform the migration only when an interruption in service is acceptable.

Prerequisites

- A cluster configured with the OpenShift SDN CNI network plugin in the network policy isolation mode.

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

- Access to the cluster as a user with the **cluster-admin** role.

- A recent backup of the etcd database is available.

- A reboot can be triggered manually for each node.

- The cluster is in a known good state, without any errors.

- On all cloud platforms after updating software, a security group rule must be in place to allow UDP packets on port **6081** for all nodes.

Procedure

1. To backup the configuration for the cluster network, enter the following command:

   ```
   $ oc get Network.config.openshift.io cluster -o yaml > cluster-openshift-sdn.yaml
   ```

2. To prepare all the nodes for the migration, set the **migration** field on the Cluster Network Operator configuration object by entering the following command:

   ```
   $ oc patch Network.operator.openshift.io cluster --type='merge' \
     --patch '{ "spec": { "migration": { "networkType": "OVNKubernetes" } } }'
   ```

   > **NOTE**
   >
   > This step does not deploy OVN-Kubernetes immediately. Instead, specifying the **migration** field triggers the Machine Config Operator (MCO) to apply new machine configs to all the nodes in the cluster in preparation for the OVN-Kubernetes deployment.

3. Optional: You can disable automatic migration of several OpenShift SDN capabilities to the OVN-Kubernetes equivalents:

   - Egress IPs

   - Egress firewall

   - Multicast

   To disable automatic migration of the configuration for any of the previously noted OpenShift SDN features, specify the following keys:

   ```
   $ oc patch Network.operator.openshift.io cluster --type='merge' \
     --patch '{
       "spec": {
         "migration": {
           "networkType": "OVNKubernetes",
           "features": {
             "egressIP": <bool>,
             "egressFirewall": <bool>,
             "multicast": <bool>
           }
         }
       }
     }'
   ```

   where:

   bool: Specifies whether to enable migration of the feature. The default is **true**.

4. Optional: You can customize the following settings for OVN-Kubernetes to meet your network infrastructure requirements:

- Maximum transmission unit (MTU). Consider the following before customizing the MTU for this optional step:

  - If you use the default MTU, and you want to keep the default MTU during migration, this step can be ignored.

  - If you used a custom MTU, and you want to keep the custom MTU during migration, you must declare the custom MTU value in this step.

  - This step does not work if you want to change the MTU value during migration. Instead, you must first follow the instructions for "Changing the cluster MTU". You can then keep the custom MTU value by performing this procedure and declaring the custom MTU value in this step.

  > **NOTE**
  >
  > OpenShift-SDN and OVN-Kubernetes have different overlay overhead. MTU values should be selected by following the guidelines found on the "MTU value selection" page.

- Geneve (Generic Network Virtualization Encapsulation) overlay network port

- OVN-Kubernetes IPv4 internal subnet

- OVN-Kubernetes IPv6 internal subnet

To customize either of the previously noted settings, enter and customize the following command. If you do not need to change the default value, omit the key from the patch.

```
$ oc patch Network.operator.openshift.io cluster --type=merge \
  --patch '{
   "spec":{
    "defaultNetwork":{
     "ovnKubernetesConfig":{
      "mtu":<mtu>,
      "genevePort":<port>,
      "v4InternalSubnet":"<ipv4_subnet>",
      "v6InternalSubnet":"<ipv6_subnet>"
   }}}}'
```

where:

mtu

The MTU for the Geneve overlay network. This value is normally configured automatically, but if the nodes in your cluster do not all use the same MTU, then you must set this explicitly to **100** less than the smallest node MTU value.

port

The UDP port for the Geneve overlay network. If a value is not specified, the default is **6081**. The port cannot be the same as the VXLAN port that is used by OpenShift SDN. The default value for the VXLAN port is **4789**.

ipv4_subnet

An IPv4 address range for internal use by OVN-Kubernetes. You must ensure that the IP address range does not overlap with any other subnet used by your OpenShift Container Platform installation. The IP address range must be larger than the maximum number of

nodes that can be added to the cluster. The default value is **100.64.0.0/16**.

ipv6_subnet

An IPv6 address range for internal use by OVN-Kubernetes. You must ensure that the IP address range does not overlap with any other subnet used by your OpenShift Container Platform installation. The IP address range must be larger than the maximum number of nodes that can be added to the cluster. The default value is **fd98::/48**.

Example patch command to update mtu field

```
$ oc patch Network.operator.openshift.io cluster --type=merge \
  --patch '{
   "spec":{
    "defaultNetwork":{
     "ovnKubernetesConfig":{
       "mtu":1200
    }}}}'
```

5. As the MCO updates machines in each machine config pool, it reboots each node one by one. You must wait until all the nodes are updated. Check the machine config pool status by entering the following command:

```
$ oc get mcp
```

A successfully updated node has the following status: **UPDATED=true**, **UPDATING=false**, **DEGRADED=false**.

> **NOTE**
>
> By default, the MCO updates one machine per pool at a time, causing the total time the migration takes to increase with the size of the cluster.

6. Confirm the status of the new machine configuration on the hosts:

   a. To list the machine configuration state and the name of the applied machine configuration, enter the following command:

   ```
   $ oc describe node | egrep "hostname|machineconfig"
   ```

   Example output

   ```
   kubernetes.io/hostname=master-0
   machineconfiguration.openshift.io/currentConfig: rendered-master-
   c53e221d9d24e1c8bb6ee89dd3d8ad7b
   machineconfiguration.openshift.io/desiredConfig: rendered-master-
   c53e221d9d24e1c8bb6ee89dd3d8ad7b
   machineconfiguration.openshift.io/reason:
   machineconfiguration.openshift.io/state: Done
   ```

   Verify that the following statements are true:

   - The value of **machineconfiguration.openshift.io/state** field is **Done**.

- The value of the **machineconfiguration.openshift.io/currentConfig** field is equal to the value of the **machineconfiguration.openshift.io/desiredConfig** field.

b. To confirm that the machine config is correct, enter the following command:

```
$ oc get machineconfig <config_name> -o yaml | grep ExecStart
```

where **<config_name>** is the name of the machine config from the **machineconfiguration.openshift.io/currentConfig** field.

The machine config must include the following update to the systemd configuration:

```
ExecStart=/usr/local/bin/configure-ovs.sh OVNKubernetes
```

c. If a node is stuck in the **NotReady** state, investigate the machine config daemon pod logs and resolve any errors.

   i. To list the pods, enter the following command:

   ```
   $ oc get pod -n openshift-machine-config-operator
   ```

   Example output

   ```
   NAME                                      READY  STATUS   RESTARTS  AGE
   machine-config-controller-75f756f89d-sjp8b  1/1    Running  0          37m
   machine-config-daemon-5cf4b                 2/2    Running  0         43h
   machine-config-daemon-7wzcd                 2/2    Running  0         43h
   machine-config-daemon-fc946                 2/2    Running  0         43h
   machine-config-daemon-g2v28                 2/2    Running  0         43h
   machine-config-daemon-gcl4f                 2/2    Running  0         43h
   machine-config-daemon-l5tnv                 2/2    Running  0         43h
   machine-config-operator-79d9c55d5-hth92     1/1    Running  0          37m
   machine-config-server-bsc8h                 1/1    Running  0         43h
   machine-config-server-hklrm                 1/1    Running  0         43h
   machine-config-server-k9rtx                 1/1    Running  0         43h
   ```

   The names for the config daemon pods are in the following format: **machine-config-daemon-<seq>**. The **<seq>** value is a random five character alphanumeric sequence.

   ii. Display the pod log for the first machine config daemon pod shown in the previous output by enter the following command:

   ```
   $ oc logs <pod> -n openshift-machine-config-operator
   ```

   where **pod** is the name of a machine config daemon pod.

   iii. Resolve any errors in the logs shown by the output from the previous command.

7. To start the migration, configure the OVN-Kubernetes network plugin by using one of the following commands:

- To specify the network provider without changing the cluster network IP address block, enter the following command:

```
$ oc patch Network.config.openshift.io cluster \
  --type='merge' --patch '{ "spec": { "networkType": "OVNKubernetes" } }'
```

- To specify a different cluster network IP address block, enter the following command:

```
$ oc patch Network.config.openshift.io cluster \
  --type='merge' --patch '{
    "spec": {
      "clusterNetwork": [
        {
          "cidr": "<cidr>",
          "hostPrefix": <prefix>
        }
      ],
      "networkType": "OVNKubernetes"
    }
  }'
```

where **cidr** is a CIDR block and **prefix** is the slice of the CIDR block apportioned to each node in your cluster. You cannot use any CIDR block that overlaps with the **100.64.0.0/16** CIDR block because the OVN-Kubernetes network provider uses this block internally.

> **IMPORTANT**
>
> You cannot change the service network address block during the migration.

8. Verify that the Multus daemon set rollout is complete before continuing with subsequent steps:

```
$ oc -n openshift-multus rollout status daemonset/multus
```

The name of the Multus pods is in the form of **multus-<xxxxx>** where **<xxxxx>** is a random sequence of letters. It might take several moments for the pods to restart.

Example output

```
Waiting for daemon set "multus" rollout to finish: 1 out of 6 new pods have been updated...
...
Waiting for daemon set "multus" rollout to finish: 5 of 6 updated pods are available...
daemon set "multus" successfully rolled out
```

9. To complete changing the network plugin, reboot each node in your cluster. You can reboot the nodes in your cluster with either of the following approaches:

- With the **oc rsh** command, you can use a bash script similar to the following:

```
#!/bin/bash
readarray -t POD_NODES <<< "$(oc get pod -n openshift-machine-config-operator -o wide| grep daemon|awk '{print $1" "$7}')"

for i in "${POD_NODES[@]}"
do
  read -r POD NODE <<< "$i"
  until oc rsh -n openshift-machine-config-operator "$POD" chroot /rootfs shutdown -r +1
    do
```

```
      echo "cannot reboot node $NODE, retry" && sleep 3
    done
  done
```

- With the **ssh** command, you can use a bash script similar to the following. The script assumes that you have configured sudo to not prompt for a password.

```
#!/bin/bash

for ip in $(oc get nodes  -o jsonpath='{.items[*].status.addresses[?
(@.type=="InternalIP")].address}')
do
  echo "reboot node $ip"
  ssh -o StrictHostKeyChecking=no core@$ip sudo shutdown -r -t 3
done
```

10. Confirm that the migration succeeded:

    a. To confirm that the network plugin is OVN-Kubernetes, enter the following command. The value of **status.networkType** must be **OVNKubernetes**.

       ```
       $ oc get network.config/cluster -o jsonpath='{.status.networkType}{"\n"}'
       ```

    b. To confirm that the cluster nodes are in the **Ready** state, enter the following command:

       ```
       $ oc get nodes
       ```

    c. To confirm that your pods are not in an error state, enter the following command:

       ```
       $ oc get pods --all-namespaces -o wide --sort-by='{.spec.nodeName}'
       ```

       If pods on a node are in an error state, reboot that node.

    d. To confirm that all of the cluster Operators are not in an abnormal state, enter the following command:

       ```
       $ oc get co
       ```

       The status of every cluster Operator must be the following: **AVAILABLE="True"**, **PROGRESSING="False"**, **DEGRADED="False"**. If a cluster Operator is not available or degraded, check the logs for the cluster Operator for more information.

11. Complete the following steps only if the migration succeeds and your cluster is in a good state:

    a. To remove the migration configuration from the CNO configuration object, enter the following command:

       ```
       $ oc patch Network.operator.openshift.io cluster --type='merge' \
         --patch '{ "spec": { "migration": null } }'
       ```

    b. To remove custom configuration for the OpenShift SDN network provider, enter the following command:

```
$ oc patch Network.operator.openshift.io cluster --type='merge' \
  --patch '{ "spec": { "defaultNetwork": { "openshiftSDNConfig": null } } }'
```

c. To remove the OpenShift SDN network provider namespace, enter the following command:

```
$ oc delete namespace openshift-sdn
```

27.5.3. Additional resources

- Configuration parameters for the OVN-Kubernetes network plugin

- Backing up etcd

- About network policy

- Changing the cluster MTU

- MTU value selection

- OVN-Kubernetes capabilities

  - Configuring an egress IP address

  - Configuring an egress firewall for a project

  - Enabling multicast for a project

- OpenShift SDN capabilities

  - Configuring egress IPs for a project

  - Configuring an egress firewall for a project

  - Enabling multicast for a project

- Network [operator.openshift.io/v1]

27.6. ROLLING BACK TO THE OPENSHIFT SDN NETWORK PROVIDER

As a cluster administrator, you can rollback to the OpenShift SDN network plugin from the OVN-Kubernetes network plugin if the migration to OVN-Kubernetes is unsuccessful.

27.6.1. Migrating to the OpenShift SDN network plugin

As a cluster administrator, you can migrate to the OpenShift SDN Container Network Interface (CNI) network plugin. During the migration you must reboot every node in your cluster.
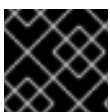
> **IMPORTANT**
>
> Rollback to OpenShift SDN if the migration to OVN-Kubernetes fails.

Prerequisites

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

- Access to the cluster as a user with the **cluster-admin** role.

- A cluster installed on infrastructure configured with the OVN-Kubernetes network plugin.

- A recent backup of the etcd database is available.

- A reboot can be triggered manually for each node.

- The cluster is in a known good state, without any errors.

Procedure

1. Stop all of the machine configuration pools managed by the Machine Config Operator (MCO):

   - Stop the master configuration pool:

     ```
     $ oc patch MachineConfigPool master --type='merge' --patch \
       '{ "spec": { "paused": true } }'
     ```

   - Stop the worker machine configuration pool:

     ```
     $ oc patch MachineConfigPool worker --type='merge' --patch \
       '{ "spec":{ "paused": true } }'
     ```

2. To prepare for the migration, set the migration field to **null** by entering the following command:

   ```
   $ oc patch Network.operator.openshift.io cluster --type='merge' \
     --patch '{ "spec": { "migration": null } }'
   ```

3. To start the migration, set the network plugin back to OpenShift SDN by entering the following commands:

   ```
   $ oc patch Network.operator.openshift.io cluster --type='merge' \
     --patch '{ "spec": { "migration": { "networkType": "OpenShiftSDN" } } }'
   ```

   ```
   $ oc patch Network.config.openshift.io cluster --type='merge' \
     --patch '{ "spec": { "networkType": "OpenShiftSDN" } }'
   ```

4. Optional: You can disable automatic migration of several OVN-Kubernetes capabilities to the OpenShift SDN equivalents:

   - Egress IPs

   - Egress firewall

   - Multicast

   To disable automatic migration of the configuration for any of the previously noted OpenShift SDN features, specify the following keys:

   ```
   $ oc patch Network.operator.openshift.io cluster --type='merge' \
     --patch '{
       "spec": {
         "migration": {
           "networkType": "OpenShiftSDN",
   ```

```
        "features": {
          "egressIP": <bool>,
          "egressFirewall": <bool>,
          "multicast": <bool>
        }
      }
    }
  }'
```

where:

bool: Specifies whether to enable migration of the feature. The default is **true**.

5. Optional: You can customize the following settings for OpenShift SDN to meet your network infrastructure requirements:

   - Maximum transmission unit (MTU)

   - VXLAN port

   To customize either or both of the previously noted settings, customize and enter the following command. If you do not need to change the default value, omit the key from the patch.

   ```
   $ oc patch Network.operator.openshift.io cluster --type=merge \
     --patch '{
       "spec":{
         "defaultNetwork":{
           "openshiftSDNConfig":{
             "mtu":<mtu>,
             "vxlanPort":<port>
       }}}}'
   ```

   mtu

   The MTU for the VXLAN overlay network. This value is normally configured automatically, but if the nodes in your cluster do not all use the same MTU, then you must set this explicitly to **50** less than the smallest node MTU value.

   port

   The UDP port for the VXLAN overlay network. If a value is not specified, the default is **4789**. The port cannot be the same as the Geneve port that is used by OVN-Kubernetes. The default value for the Geneve port is **6081**.

   ### Example patch command

   ```
   $ oc patch Network.operator.openshift.io cluster --type=merge \
     --patch '{
       "spec":{
         "defaultNetwork":{
           "openshiftSDNConfig":{
             "mtu":1200
       }}}}'
   ```

6. Reboot each node in your cluster. You can reboot the nodes in your cluster with either of the following approaches:

   - With the **oc rsh** command, you can use a bash script similar to the following:

```
#!/bin/bash
readarray -t POD_NODES <<< "$(oc get pod -n openshift-machine-config-operator -o
wide| grep daemon|awk '{print $1" "$7}')"

for i in "${POD_NODES[@]}"
do
  read -r POD NODE <<< "$i"
  until oc rsh -n openshift-machine-config-operator "$POD" chroot /rootfs shutdown -r +1
    do
      echo "cannot reboot node $NODE, retry" && sleep 3
    done
done
```

- With the **ssh** command, you can use a bash script similar to the following. The script assumes that you have configured sudo to not prompt for a password.

```
#!/bin/bash

for ip in $(oc get nodes  -o jsonpath='{.items[*].status.addresses[?
(@.type=="InternalIP")].address}')
do
  echo "reboot node $ip"
  ssh -o StrictHostKeyChecking=no core@$ip sudo shutdown -r -t 3
done
```

7. Wait until the Multus daemon set rollout completes. Run the following command to see your rollout status:

```
$ oc -n openshift-multus rollout status daemonset/multus
```

The name of the Multus pods is in the form of **multus-<xxxxx>** where **<xxxxx>** is a random sequence of letters. It might take several moments for the pods to restart.

Example output

```
Waiting for daemon set "multus" rollout to finish: 1 out of 6 new pods have been updated...
...
Waiting for daemon set "multus" rollout to finish: 5 of 6 updated pods are available...
daemon set "multus" successfully rolled out
```

8. After the nodes in your cluster have rebooted and the multus pods are rolled out, start all of the machine configuration pools by running the following commands::

- Start the master configuration pool:

```
$ oc patch MachineConfigPool master --type='merge' --patch \
  '{ "spec": { "paused": false } }'
```

- Start the worker configuration pool:

```
$ oc patch MachineConfigPool worker --type='merge' --patch \
  '{ "spec": { "paused": false } }'
```

As the MCO updates machines in each config pool, it reboots each node.

By default the MCO updates a single machine per pool at a time, so the time that the migration requires to complete grows with the size of the cluster.

9. Confirm the status of the new machine configuration on the hosts:

   a. To list the machine configuration state and the name of the applied machine configuration, enter the following command:

      ```
      $ oc describe node | egrep "hostname|machineconfig"
      ```

      Example output

      ```
      kubernetes.io/hostname=master-0
      machineconfiguration.openshift.io/currentConfig: rendered-master-
      c53e221d9d24e1c8bb6ee89dd3d8ad7b
      machineconfiguration.openshift.io/desiredConfig: rendered-master-
      c53e221d9d24e1c8bb6ee89dd3d8ad7b
      machineconfiguration.openshift.io/reason:
      machineconfiguration.openshift.io/state: Done
      ```

      Verify that the following statements are true:

      - The value of **machineconfiguration.openshift.io/state** field is **Done**.

      - The value of the **machineconfiguration.openshift.io/currentConfig** field is equal to the value of the **machineconfiguration.openshift.io/desiredConfig** field.

   b. To confirm that the machine config is correct, enter the following command:

      ```
      $ oc get machineconfig <config_name> -o yaml
      ```

      where **<config_name>** is the name of the machine config from the **machineconfiguration.openshift.io/currentConfig** field.

10. Confirm that the migration succeeded:

   a. To confirm that the network plugin is OpenShift SDN, enter the following command. The value of **status.networkType** must be **OpenShiftSDN**.

      ```
      $ oc get network.config/cluster -o jsonpath='{.status.networkType}{"\n"}'
      ```

   b. To confirm that the cluster nodes are in the **Ready** state, enter the following command:

      ```
      $ oc get nodes
      ```

   c. If a node is stuck in the **NotReady** state, investigate the machine config daemon pod logs and resolve any errors.

      i. To list the pods, enter the following command:

         ```
         $ oc get pod -n openshift-machine-config-operator
         ```

         Example output

```
NAME                                      READY   STATUS    RESTARTS   AGE
machine-config-controller-75f756f89d-sjp8b  1/1    Running   0          37m
machine-config-daemon-5cf4b                 2/2    Running   0          43h
machine-config-daemon-7wzcd                 2/2    Running   0          43h
machine-config-daemon-fc946                 2/2    Running   0          43h
machine-config-daemon-g2v28                 2/2    Running   0          43h
machine-config-daemon-gcl4f                 2/2    Running   0          43h
machine-config-daemon-l5tnv                 2/2    Running   0          43h
machine-config-operator-79d9c55d5-hth92     1/1    Running   0          37m
machine-config-server-bsc8h                 1/1    Running   0          43h
machine-config-server-hklrm                 1/1    Running   0          43h
machine-config-server-k9rtx                 1/1    Running   0          43h
```

The names for the config daemon pods are in the following format: **machine-config-daemon-<seq>**. The **<seq>** value is a random five character alphanumeric sequence.

ii. To display the pod log for each machine config daemon pod shown in the previous output, enter the following command:

```
$ oc logs <pod> -n openshift-machine-config-operator
```

where **pod** is the name of a machine config daemon pod.

iii. Resolve any errors in the logs shown by the output from the previous command.

d. To confirm that your pods are not in an error state, enter the following command:

```
$ oc get pods --all-namespaces -o wide --sort-by='{.spec.nodeName}'
```

If pods on a node are in an error state, reboot that node.

11. Complete the following steps only if the migration succeeds and your cluster is in a good state:

a. To remove the migration configuration from the Cluster Network Operator configuration object, enter the following command:

```
$ oc patch Network.operator.openshift.io cluster --type='merge' \
  --patch '{ "spec": { "migration": null } }'
```

b. To remove the OVN-Kubernetes configuration, enter the following command:

```
$ oc patch Network.operator.openshift.io cluster --type='merge' \
  --patch '{ "spec": { "defaultNetwork": { "ovnKubernetesConfig":null } } }'
```

c. To remove the OVN-Kubernetes network provider namespace, enter the following command:

```
$ oc delete namespace openshift-ovn-kubernetes
```

27.7. MIGRATING FROM THE KURYR NETWORK PLUGIN TO THE OVN-KUBERNETES NETWORK PLUGIN

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

As the administrator of a cluster that runs on Red Hat OpenStack Platform (RHOSP), you can migrate to the OVN-Kubernetes network plugin from the Kuryr SDN network plugin.

To learn more about OVN-Kubernetes, read About the OVN-Kubernetes network plugin .

27.7.1. Migration to the OVN-Kubernetes network provider

You can manually migrate a cluster that runs on Red Hat OpenStack Platform (RHOSP) to the OVN-Kubernetes network provider.

> **IMPORTANT**
>
> Migration to OVN-Kubernetes is a one-way process. During migration, your cluster will be unreachable for a brief time.

27.7.1.1. Considerations when migrating to the OVN-Kubernetes network provider

Kubernetes namespaces are kept by Kuryr in separate RHOSP networking service (Neutron) subnets. Those subnets and the IP addresses that are assigned to individual pods are not preserved during the migration.

27.7.1.2. How the migration process works

The following table summarizes the migration process by relating the steps that you perform with the actions that your cluster and Operators take.

Table 27.9. The Kuryr to OVN-Kubernetes migration process

User-initiated steps	Migration activity
Set the **migration** field of the **Network.operator.openshift.io** custom resource (CR) named **cluster** to **OVNKubernetes**. Verify that the value of the **migration** field prints the **null** value before setting it to another value.	**Cluster Network Operator (CNO)** Updates the status of the **Network.config.openshift.io** CR named **cluster** accordingly. **Machine Config Operator (MCO)** Deploys an update to the systemd configuration that is required by OVN-Kubernetes. By default, the MCO updates a single machine per pool at a time. As a result, large clusters have longer migration times.

User-initiated steps	Migration activity

Update the **networkType** field of the **Network.config.openshift.io** CR.	**CNO**
	Performs the following actions:
	
	    • Destroys the Kuryr control plane pods: Kuryr CNIs and the Kuryr controller.
	
	    • Deploys the OVN-Kubernetes control plane pods.
	
	    • Updates the Multus objects to reflect the new network plugin.

Reboot each node in the cluster.	**Cluster** As nodes reboot, the cluster assigns IP addresses to pods on the OVN-Kubernetes cluster network.
Clean up remaining resources Kuryr controlled.	**Cluster** Holds RHOSP resources that need to be freed, as well as OpenShift Container Platform resources to configure.

27.7.2. Migrating to the OVN-Kubernetes network plugin

As a cluster administrator, you can change the network plugin for your cluster to OVN-Kubernetes.

> **IMPORTANT**
>
> During the migration, you must reboot every node in your cluster. Your cluster is unavailable and workloads might be interrupted. Perform the migration only if an interruption in service is acceptable.

Prerequisites

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

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

- You have a recent backup of the etcd database is available.

- You can manually reboot each node.

- The cluster you plan to migrate is in a known good state, without any errors.

- You installed the Python interpreter.

- You installed the **openstacksdk** python package.

- You installed the **openstack** CLI tool.

- You have access to the underlying RHOSP cloud.

Procedure

1. Back up the configuration for the cluster network by running the following command:

   ```
   $ oc get Network.config.openshift.io cluster -o yaml > cluster-kuryr.yaml
   ```

2. To set the **CLUSTERID** variable, run the following command:

   ```
   $ CLUSTERID=$(oc get infrastructure.config.openshift.io cluster -
   o=jsonpath='{.status.infrastructureName}')
   ```

3. To prepare all the nodes for the migration, set the **migration** field on the Cluster Network Operator configuration object by running the following command:

   ```
   $ oc patch Network.operator.openshift.io cluster --type='merge' \
     --patch '{ "spec": { "migration": { "networkType": "OVNKubernetes" } } }'
   ```

   > **NOTE**
   >
   > This step does not deploy OVN-Kubernetes immediately. Specifying the **migration** field triggers the Machine Config Operator (MCO) to apply new machine configs to all the nodes in the cluster. This prepares the cluster for the OVN-Kubernetes deployment.

4. Optional: Customize the following settings for OVN-Kubernetes for your network infrastructure requirements:

   - Maximum transmission unit (MTU)

   - Geneve (Generic Network Virtualization Encapsulation) overlay network port

   - OVN-Kubernetes IPv4 internal subnet

   - OVN-Kubernetes IPv6 internal subnet

   To customize these settings, enter and customize the following command:

   ```
   $ oc patch Network.operator.openshift.io cluster --type=merge \
     --patch '{
      "spec":{
       "defaultNetwork":{
        "ovnKubernetesConfig":{
         "mtu":<mtu>,
         "genevePort":<port>,
         "v4InternalSubnet":"<ipv4_subnet>",
         "v6InternalSubnet":"<ipv6_subnet>"
     }}}}'
   ```

   where:

   mtu

Specifies the MTU for the Geneve overlay network. This value is normally configured automatically, but if the nodes in your cluster do not all use the same MTU, then you must set this explicitly to **100** less than the smallest node MTU value.

port

Specifies the UDP port for the Geneve overlay network. If a value is not specified, the default is **6081**. The port cannot be the same as the VXLAN port that is used by Kuryr. The default value for the VXLAN port is **4789**.

ipv4_subnet

Specifies an IPv4 address range for internal use by OVN-Kubernetes. You must ensure that the IP address range does not overlap with any other subnet used by your OpenShift Container Platform installation. The IP address range must be larger than the maximum number of nodes that can be added to the cluster. The default value is **100.64.0.0/16**.

ipv6_subnet

Specifies an IPv6 address range for internal use by OVN-Kubernetes. You must ensure that the IP address range does not overlap with any other subnet used by your OpenShift Container Platform installation. The IP address range must be larger than the maximum number of nodes that can be added to the cluster. The default value is **fd98::/48**.

If you do not need to change the default value, omit the key from the patch.

Example patch command to update mtu field

```
$ oc patch Network.operator.openshift.io cluster --type=merge \
  --patch '{
    "spec":{
      "defaultNetwork":{
        "ovnKubernetesConfig":{
          "mtu":1200
    }}}}'
```

5. Check the machine config pool status by entering the following command:

```
$ oc get mcp
```

While the MCO updates machines in each machine config pool, it reboots each node one by one. You must wait until all the nodes are updated before continuing.

A successfully updated node has the following status: **UPDATED=true**, **UPDATING=false**, **DEGRADED=false**.

> **NOTE**
>
> By default, the MCO updates one machine per pool at a time. Large clusters take more time to migrate than small clusters.

6. Confirm the status of the new machine configuration on the hosts:

   a. To list the machine configuration state and the name of the applied machine configuration, enter the following command:

   ```
   $ oc describe node | egrep "hostname|machineconfig"
   ```

Example output

```
kubernetes.io/hostname=master-0
machineconfiguration.openshift.io/currentConfig: rendered-master-
c53e221d9d24e1c8bb6ee89dd3d8ad7b 1
machineconfiguration.openshift.io/desiredConfig: rendered-master-
c53e221d9d24e1c8bb6ee89dd3d8ad7b 2
machineconfiguration.openshift.io/reason:
machineconfiguration.openshift.io/state: Done
```

b. Review the output from the previous step. The following statements must be true:

- The value of **machineconfiguration.openshift.io/state** field is **Done**.

- The value of the **machineconfiguration.openshift.io/currentConfig** field is equal to the value of the **machineconfiguration.openshift.io/desiredConfig** field.

c. To confirm that the machine config is correct, enter the following command:

```
$ oc get machineconfig <config_name> -o yaml | grep ExecStart
```

where:

<config_name>

Specifies the name of the machine config from the **machineconfiguration.openshift.io/currentConfig** field.
The machine config must include the following update to the systemd configuration:

Example output

```
ExecStart=/usr/local/bin/configure-ovs.sh OVNKubernetes
```

d. If a node is stuck in the **NotReady** state, investigate the machine config daemon pod logs and resolve any errors:

i. To list the pods, enter the following command:

```
$ oc get pod -n openshift-machine-config-operator
```

Example output

```
NAME                                 READY   STATUS    RESTARTS   AGE
machine-config-controller-75f756f89d-sjp8b   1/1     Running   0          37m
machine-config-daemon-5cf4b                  2/2     Running   0          43h
machine-config-daemon-7wzcd                  2/2     Running   0          43h
machine-config-daemon-fc946                  2/2     Running   0          43h
machine-config-daemon-g2v28                  2/2     Running   0          43h
machine-config-daemon-gcl4f                  2/2     Running   0          43h
machine-config-daemon-l5tnv                  2/2     Running   0          43h
machine-config-operator-79d9c55d5-hth92      1/1     Running   0          37m
machine-config-server-bsc8h                  1/1     Running   0          43h
machine-config-server-hklrm                  1/1     Running   0          43h
machine-config-server-k9rtx                  1/1     Running   0          43h
```

The names for the config daemon pods are in the following format: **machine-config-daemon-<seq>**. The **<seq>** value is a random five character alphanumeric sequence.

ii. Display the pod log for the first machine config daemon pod shown in the previous output by enter the following command:

```
$ oc logs <pod> -n openshift-machine-config-operator
```

where:

<pod>

Specifies the name of a machine config daemon pod.

iii. Resolve any errors in the logs shown by the output from the previous command.

7. To start the migration, configure the OVN-Kubernetes network plugin by using one of the following commands:

- To specify the network provider without changing the cluster network IP address block, enter the following command:

```
$ oc patch Network.config.openshift.io cluster \
  --type='merge' --patch '{ "spec": { "networkType": "OVNKubernetes" } }'
```

- To specify a different cluster network IP address block, enter the following command:

```
$ oc patch Network.config.openshift.io cluster \
  --type='merge' --patch '{
  "spec": {
    "clusterNetwork": [
     {
       "cidr": "<cidr>",
       "hostPrefix": "<prefix>"
     }
    ]
    "networkType": "OVNKubernetes"
  }
}'
```

where:

<cidr>

Specifies a CIDR block.

<prefix>

Specifies a slice of the CIDR block that is apportioned to each node in your cluster.

> **IMPORTANT**
>
> You cannot change the service network address block during the migration.
>
> You cannot use any CIDR block that overlaps with the **100.64.0.0/16** CIDR block because the OVN-Kubernetes network provider uses this block internally.

8. Verify that the Multus daemon set rollout is complete by entering the following command:

```
$ oc -n openshift-multus rollout status daemonset/multus
```

The name of the Multus pods is in the form of **multus-<xxxxx>**, where **<xxxxx>** is a random sequence of letters. It might take several moments for the pods to restart.

Example output

```
Waiting for daemon set "multus" rollout to finish: 1 out of 6 new pods have been updated...
...
Waiting for daemon set "multus" rollout to finish: 5 of 6 updated pods are available...
daemon set "multus" successfully rolled out
```

9. To complete the migration, reboot each node in your cluster. For example, you can use a bash script similar to the following example. The script assumes that you can connect to each host by using **ssh** and that you have configured **sudo** to not prompt for a password:

```bash
#!/bin/bash

for ip in $(oc get nodes  -o jsonpath='{.items[*].status.addresses[?(@.type=="InternalIP")].address}')
do
   echo "reboot node $ip"
   ssh -o StrictHostKeyChecking=no core@$ip sudo shutdown -r -t 3
done
```

> **NOTE**
>
> If SSH access is not available, you can use the **openstack** command:
>
> ```
> $ for name in $(openstack server list --name ${CLUSTERID}\* -f value -c Name); do openstack server reboot $name; done
> ```
>
> Alternatively, you might be able to reboot each node through the management portal for your infrastructure provider. Otherwise, contact the appropriate authority to either gain access to the virtual machines through SSH or the management portal and OpenStack client.

Verification

1. Confirm that the migration succeeded, and then remove the migration resources:

   a. To confirm that the network plugin is OVN-Kubernetes, enter the following command.

   ```
   $ oc get network.config/cluster -o jsonpath='{.status.networkType}{"\n"}'
   ```

   The value of **status.networkType** must be **OVNKubernetes**.

   b. To confirm that the cluster nodes are in the **Ready** state, enter the following command:

   ```
   $ oc get nodes
   ```

c. To confirm that your pods are not in an error state, enter the following command:

```
$ oc get pods --all-namespaces -o wide --sort-by='{.spec.nodeName}'
```

If pods on a node are in an error state, reboot that node.

d. To confirm that all of the cluster Operators are not in an abnormal state, enter the following command:

```
$ oc get co
```

The status of every cluster Operator must be the following: **AVAILABLE="True"**, **PROGRESSING="False"**, **DEGRADED="False"**. If a cluster Operator is not available or degraded, check the logs for the cluster Operator for more information.

> **IMPORTANT**
>
> Do not proceed if any of the previous verification steps indicate errors. You might encounter pods that have a **Terminating** state due to finalizers that are removed during clean up. They are not an error indication.

2. If the migration completed and your cluster is in a good state, remove the migration configuration from the CNO configuration object by entering the following command:

```
$ oc patch Network.operator.openshift.io cluster --type='merge' \
  --patch '{ "spec": { "migration": null } }'
```

27.7.3. Cleaning up resources after migration

After migration from the Kuryr network plugin to the OVN-Kubernetes network plugin, you must clean up the resources that Kuryr created previously.

> **NOTE**
>
> The clean up process relies on a Python virtual environment to ensure that the package versions that you use support tags for Octavia objects. You do not need a virtual environment if you are certain that your environment uses at minimum: * **openstacksdk** version 0.54.0 * **python-openstackclient** version 5.5.0 * **python-octaviaclient** version 2.3.0

Prerequisites

- You installed the OpenShift Container Platform CLI (**oc**).

- You installed a Python interpreter.

- You installed the **openstacksdk** Python package.

- You installed the **openstack** CLI.

- You have access to the underlying RHOSP cloud.

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

Procedure

1. Create a clean-up Python virtual environment:

   a. Create a temporary directory for your environment. For example:

   ```
   $ python3 -m venv /tmp/venv
   ```

   The virtual environment located in **/tmp/venv** directory is used in all clean up examples.

   b. Enter the virtual environment. For example:

   ```
   $ source /tmp/venv/bin/activate
   ```

   c. Upgrade the **pip** command in the virtual environment by running the following command:

   ```
   (venv) $ pip install pip --upgrade
   ```

   d. Install the required Python packages by running the following command:

   ```
   (venv) $ pip install openstacksdk==0.54.0 python-openstackclient==5.5.0 python-octaviaclient==2.3.0
   ```

2. In your terminal, set variables to cluster and Kuryr identifiers by running the following commands:

   a. Set the cluster ID:

   ```
   (venv) $ CLUSTERID=$(oc get infrastructure.config.openshift.io cluster -o=jsonpath='{.status.infrastructureName}')
   ```

   b. Set the cluster tag:

   ```
   (venv) $ CLUSTERTAG="openshiftClusterID=${CLUSTERID}"
   ```

   c. Set the router ID:

   ```
   (venv) $ ROUTERID=$(oc get kuryrnetwork -A --no-headers -o custom-columns=":status.routerId"|head -n 1)
   ```

3. Create a Bash function that removes finalizers from specified resources by running the following command:

   ```
   (venv) $ function REMFIN {
       local resource=$1
       local finalizer=$2
       for res in $(oc get $resource -A --template='{{range $i,$p := .items}}{{ $p.metadata.name }}|{{ $p.metadata.namespace }}{{"\n"}}{{end}}'); do
           name=${res%%|*}
           ns=${res##*|}
           yaml=$(oc get -n $ns $resource $name -o yaml)
           if echo "${yaml}" | grep -q "${finalizer}"; then
               echo "${yaml}" | grep -v  "${finalizer}" | oc replace -n $ns $resource $name -f -
   ```

```
        fi
    done
}
```

The function takes two parameters: the first parameter is name of the resource, and the second parameter is the finalizer to remove. The named resource is removed from the cluster and its definition is replaced with copied data, excluding the specified finalizer.

4. To remove Kuryr finalizers from services, enter the following command:

   ```
   (venv) $ REMFIN services kuryr.openstack.org/service-finalizer
   ```

5. To remove the Kuryr **service-subnet-gateway-ip** service, enter the following command:

   ```
   (venv) $ if $(oc get -n openshift-kuryr service service-subnet-gateway-ip &>/dev/null); then
       oc -n openshift-kuryr delete service service-subnet-gateway-ip
   fi
   ```

6. To remove all tagged RHOSP load balancers from Octavia, enter the following command:

   ```
   (venv) $ for lb in $(openstack loadbalancer list --tags $CLUSTERTAG -f value -c id); do
       openstack loadbalancer delete --cascade $lb
   done
   ```

7. To remove Kuryr finalizers from all **KuryrLoadBalancer** CRs, enter the following command:

   ```
   (venv) $ REMFIN kuryrloadbalancers.openstack.org kuryr.openstack.org/kuryrloadbalancer-finalizers
   ```

8. To remove the **openshift-kuryr** namespace, enter the following command:

   ```
   (venv) $ oc delete namespace openshift-kuryr
   ```

9. To remove the Kuryr service subnet from the router, enter the following command:

   ```
   (venv) $ openstack router remove subnet $ROUTERID ${CLUSTERID}-kuryr-service-subnet
   ```

10. To remove the Kuryr service network, enter the following command:

    ```
    (venv) $ openstack network delete ${CLUSTERID}-kuryr-service-network
    ```

11. To remove Kuryr finalizers from all pods, enter the following command:

    ```
    (venv) $ REMFIN pods kuryr.openstack.org/pod-finalizer
    ```

12. To remove Kuryr finalizers from all **KuryrPort** CRs, enter the following command:

    ```
    (venv) $ REMFIN kuryrports.openstack.org kuryr.openstack.org/kuryrport-finalizer
    ```

    This command deletes the **KuryrPort** CRs.

13. To remove Kuryr finalizers from network policies, enter the following command:

```
(venv) $ REMFIN networkpolicy kuryr.openstack.org/networkpolicy-finalizer
```

14. To remove Kuryr finalizers from remaining network policies, enter the following command:

```
(venv) $ REMFIN kuryrnetworkpolicies.openstack.org kuryr.openstack.org/networkpolicy-finalizer
```

15. To remove subports that Kuryr created from trunks, enter the following command:

```
(venv) $ read -ra trunks <<< $(python -c "import openstack; n = openstack.connect().network;
print(' '.join([x.id for x in n.trunks(any_tags='$CLUSTERTAG')]))") && \
i=0 && \
for trunk in "${trunks[@]}"; do
    i=$((i+1))
    echo "Processing trunk $trunk, ${i}/${#trunks[@]}."
    subports=()
    for subport in $(python -c "import openstack; n = openstack.connect().network; print('
'.join([x['port_id'] for x in n.get_trunk('$trunk').sub_ports if '$CLUSTERTAG' in
n.get_port(x['port_id']).tags]))"); do
        subports+=("$subport");
    done
    args=()
    for sub in "${subports[@]}" ; do
        args+=("--subport $sub")
    done
    if [ ${#args[@]} -gt 0 ]; then
        openstack network trunk unset ${args[*]} $trunk
    fi
done
```

16. To retrieve all networks and subnets from **KuryrNetwork** CRs and remove ports, router interfaces and the network itself, enter the following command:

```
(venv) $ mapfile -t kuryrnetworks < <(oc get kuryrnetwork -A --template='{{range $i,$p :=
.items}}{{ $p.status.netId }}|{{ $p.status.subnetId }}{{"\n"}}{{end}}') && \
i=0 && \
for kn in "${kuryrnetworks[@]}"; do
    i=$((i+1))
    netID=${kn%%|*}
    subnetID=${kn##*|}
    echo "Processing network $netID, ${i}/${#kuryrnetworks[@]}"
    # Remove all ports from the network.
    for port in $(python -c "import openstack; n = openstack.connect().network; print(' '.join([x.id
for x in n.ports(network_id='$netID') if x.device_owner != 'network:router_interface']))"); do
        ( openstack port delete $port ) &

        # Only allow 20 jobs in parallel.
        if [[ $(jobs -r -p | wc -l) -ge 20 ]]; then
            wait -n
        fi
    done
    wait

    # Remove the subnet from the router.
    openstack router remove subnet $ROUTERID $subnetID
```

```
    # Remove the network.
    openstack network delete $netID
done
```

17. To remove the Kuryr security group, enter the following command:

```
(venv) $ openstack security group delete ${CLUSTERID}-kuryr-pods-security-group
```

18. To remove all tagged subnet pools, enter the following command:

```
(venv) $ for subnetpool in $(openstack subnet pool list --tags $CLUSTERTAG -f value -c ID);
do
    openstack subnet pool delete $subnetpool
done
```

19. To check that all of the networks based on **KuryrNetwork** CRs were removed, enter the following command:

```
(venv) $ networks=$(oc get kuryrnetwork -A --no-headers -o custom-
columns=":status.netId") && \
for existingNet in $(openstack network list --tags $CLUSTERTAG -f value -c ID); do
    if [[ $networks =~ $existingNet ]]; then
        echo "Network still exists: $existingNet"
    fi
done
```

If the command returns any existing networks, intestigate and remove them before you continue.

20. To remove security groups that are related to network policy, enter the following command:

```
(venv) $ for sgid in $(openstack security group list -f value -c ID -c Description | grep 'Kuryr-
Kubernetes Network Policy' | cut -f 1 -d ' '); do
    openstack security group delete $sgid
done
```

21. To remove finalizers from **KuryrNetwork** CRs, enter the following command:

```
(venv) $ REMFIN kuryrnetworks.openstack.org kuryrnetwork.finalizers.kuryr.openstack.org
```

22. To remove the Kuryr router, enter the following command:

```
(venv) $ if $(python3 -c "import sys; import openstack; n = openstack.connect().network; r =
n.get_router('$ROUTERID'); sys.exit(0) if r.description != 'Created By OpenShift Installer'
else sys.exit(1)"); then
    openstack router delete $ROUTERID
fi
```

27.7.4. Additional resources

- [Configuration parameters for the OVN-Kubernetes network plugin](#)

- [Backing up etcd](#)

- [About network policy](#)

- To learn more about OVN-Kubernetes capabilities, see:

  - [Configuring an egress IP address](#)

  - [Configuring an egress firewall for a project](#)

  - [Enabling multicast for a project](#)

27.8. CONVERTING TO IPV4/IPV6 DUAL-STACK NETWORKING

As a cluster administrator, you can convert your IPv4 single-stack cluster to a dual-network cluster network that supports IPv4 and IPv6 address families. After converting to dual-stack, all newly created pods are dual-stack enabled.
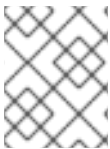
> **NOTE**
>
> A dual-stack network is supported on clusters provisioned on bare metal, VMware vSphere, IBM Power, IBM Z infrastructure, and single node OpenShift clusters.

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

27.8.1. Converting to a dual-stack cluster network

As a cluster administrator, you can convert your single-stack cluster network to a dual-stack cluster network.

> **NOTE**
>
> After converting to dual-stack networking only newly created pods are assigned IPv6 addresses. Any pods created before the conversion must be recreated to receive an IPv6 address.

Prerequisites

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

- You are logged in to the cluster with a user with **cluster-admin** privileges.

- Your cluster uses the OVN-Kubernetes network plugin.

- The cluster nodes have IPv6 addresses.

- You have configured an IPv6-enabled router based on your infrastructure.

Procedure

1. To specify IPv6 address blocks for the cluster and service networks, create a file containing the following YAML:

```
- op: add
  path: /spec/clusterNetwork/-
  value: ❶
    cidr: fd01::/48
    hostPrefix: 64
- op: add
  path: /spec/serviceNetwork/-
  value: fd02::/112 ❷
```

❶ Specify an object with the **cidr** and **hostPrefix** fields. The host prefix must be **64** or greater. The IPv6 CIDR prefix must be large enough to accommodate the specified host prefix.

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

2. To patch the cluster network configuration, enter the following command:

```
$ oc patch network.config.openshift.io cluster \
  --type='json' --patch-file <file>.yaml
```

where:

file

Specifies the name of the file you created in the previous step.

Example output

```
network.config.openshift.io/cluster patched
```

Verification

Complete the following step to verify that the cluster network recognizes the IPv6 address blocks that you specified in the previous procedure.

1. Display the network configuration:

```
$ oc describe network
```

Example output

```
Status:
  Cluster Network:
    Cidr:            10.128.0.0/14
    Host Prefix:       23
    Cidr:            fd01::/48
    Host Prefix:       64
  Cluster Network MTU:  1400
  Network Type:          OVNKubernetes
```

> Service Network:
>   172.30.0.0/16
>   fd02::/112

27.8.2. Converting to a single-stack cluster network

As a cluster administrator, you can convert your dual-stack cluster network to a single-stack cluster network.

Prerequisites

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

- You are logged in to the cluster with a user with **cluster-admin** privileges.

- Your cluster uses the OVN-Kubernetes network plugin.

- The cluster nodes have IPv6 addresses.

- You have enabled dual-stack networking.

Procedure

1. Edit the **networks.config.openshift.io** custom resource (CR) by running the following command:

   > $ oc edit networks.config.openshift.io

2. Remove the IPv6 specific configuration that you have added to the **cidr** and **hostPrefix** fields in the previous procedure.

27.9. LOGGING FOR EGRESS FIREWALL AND NETWORK POLICY RULES

As a cluster administrator, you can configure audit logging for your cluster and enable logging for one or more namespaces. OpenShift Container Platform produces audit logs for both egress firewalls and network policies.

> **NOTE**
>
> Audit logging is available for only the OVN-Kubernetes network plugin .

27.9.1. Audit logging

The OVN-Kubernetes network plugin uses Open Virtual Network (OVN) ACLs to manage egress firewalls and network policies. Audit logging exposes allow and deny ACL events.

You can configure the destination for audit logs, such as a syslog server or a UNIX domain socket. Regardless of any additional configuration, an audit log is always saved to **/var/log/ovn/acl-audit-log.log** on each OVN-Kubernetes pod in the cluster.

Audit logging is enabled per namespace by annotating the namespace with the **k8s.ovn.org/acl-logging** key as in the following example:

Example namespace annotation

```
kind: Namespace
apiVersion: v1
metadata:
  name: example1
  annotations:
    k8s.ovn.org/acl-logging: |-
      {
        "deny": "info",
        "allow": "info"
      }
```

The logging format is compatible with syslog as defined by RFC5424. The syslog facility is configurable and defaults to **local0**. An example log entry might resemble the following:

Example ACL deny log entry for a network policy

```
Defaulting container name to ovn-controller.
Use 'oc describe pod/ovnkube-node-hdb8v -n openshift-ovn-kubernetes' to see all of the containers
in this pod.
2021-06-13T19:33:11.590Z|00005|acl_log(ovn_pinctrl0)|INFO|name="verify-audit-
logging_ingressDefaultDeny", verdict=drop, severity=alert:
icmp,vlan_tci=0x0000,dl_src=0a:58:0a:80:02:39,dl_dst=0a:58:0a:80:02:37,nw_src=10.128.2.57,nw_dst=
10.128.2.55,nw_tos=0,nw_ecn=0,nw_ttl=64,icmp_type=8,icmp_code=0
2021-06-13T19:33:12.614Z|00006|acl_log(ovn_pinctrl0)|INFO|name="verify-audit-
logging_ingressDefaultDeny", verdict=drop, severity=alert:
icmp,vlan_tci=0x0000,dl_src=0a:58:0a:80:02:39,dl_dst=0a:58:0a:80:02:37,nw_src=10.128.2.57,nw_dst=
10.128.2.55,nw_tos=0,nw_ecn=0,nw_ttl=64,icmp_type=8,icmp_code=0
2021-06-13T19:44:10.037Z|00007|acl_log(ovn_pinctrl0)|INFO|name="verify-audit-logging_allow-
from-same-namespace_0", verdict=allow, severity=alert:
icmp,vlan_tci=0x0000,dl_src=0a:58:0a:80:02:3b,dl_dst=0a:58:0a:80:02:3a,nw_src=10.128.2.59,nw_dst=
10.128.2.58,nw_tos=0,nw_ecn=0,nw_ttl=64,icmp_type=8,icmp_code=0
2021-06-13T19:44:11.037Z|00008|acl_log(ovn_pinctrl0)|INFO|name="verify-audit-logging_allow-
from-same-namespace_0", verdict=allow, severity=alert:
icmp,vlan_tci=0x0000,dl_src=0a:58:0a:80:02:3b,dl_dst=0a:58:0a:80:02:3a,nw_src=10.128.2.59,nw_dst=
10.128.2.58,nw_tos=0,nw_ecn=0,nw_ttl=64,icmp_type=8,icmp_code=0
```

The following table describes namespace annotation values:

Table 27.10. Audit logging namespace annotation

Annotation	Value

k8s.ovn.org/acl-logging	You must specify at least one of **allow**, **deny**, or both to enable audit logging for a namespace.
	
	deny
	    Optional: Specify **alert**, **warning**, **notice**, **info**, or **debug**.
	allow
	    Optional: Specify **alert**, **warning**, **notice**, **info**, or **debug**.

27.9.2. Audit configuration

The configuration for audit logging is specified as part of the OVN-Kubernetes cluster network provider configuration. The following YAML illustrates the default values for the audit logging:

Audit logging configuration

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  defaultNetwork:
    ovnKubernetesConfig:
      policyAuditConfig:
        destination: "null"
        maxFileSize: 50
        rateLimit: 20
        syslogFacility: local0
```

The following table describes the configuration fields for audit logging.

Table 27.11. **policyAuditConfig** object

Field	Type	Description
rateLimit	integer	The maximum number of messages to generate every second per node. The default value is **20** messages per second.
maxFileSize	integer	The maximum size for the audit log in bytes. The default value is **50000000** or 50 MB.
maxLogFiles	integer	The maximum number of log files that are retained.
destination	string	One of the following additional audit log targets: **libc** The libc **syslog()** function of the journald process on the host. **udp:<host>:<port>** A syslog server. Replace **<host>:<port>** with the host and port of the syslog server. **unix:<file>** A Unix Domain Socket file specified by **<file>**. **null** Do not send the audit logs to any additional target.
syslogFacility	string	The syslog facility, such as **kern**, as defined by RFC5424. The default value is **local0**.

27.9.3. Configuring egress firewall and network policy auditing for a cluster

As a cluster administrator, you can customize audit logging for your cluster.

Prerequisites

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

- Log in to the cluster with a user with **cluster-admin** privileges.

Procedure

- To customize the audit logging configuration, enter the following command:

  ```
  $ oc edit network.operator.openshift.io/cluster
  ```

  ### TIP

  You can alternatively customize and apply the following YAML to configure audit logging:

  ```yaml
  apiVersion: operator.openshift.io/v1
  kind: Network
  metadata:
    name: cluster
  spec:
   defaultNetwork:
     ovnKubernetesConfig:
       policyAuditConfig:
         destination: "null"
         maxFileSize: 50
         rateLimit: 20
         syslogFacility: local0
  ```

Verification

1. To create a namespace with network policies complete the following steps:

   a. Create a namespace for verification:

      ```
      $ cat <<EOF| oc create -f -
      kind: Namespace
      apiVersion: v1
      metadata:
        name: verify-audit-logging
        annotations:
          k8s.ovn.org/acl-logging: '{ "deny": "alert", "allow": "alert" }'
      EOF
      ```

   ### Example output

      ```
      namespace/verify-audit-logging created
      ```

   b. Create network policies for the namespace:

      ```
      $ cat <<EOF| oc create -n verify-audit-logging -f -
      ```

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: deny-all
spec:
  podSelector:
    matchLabels:
  policyTypes:
  - Ingress
  - Egress
---
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-same-namespace
  namespace: verify-audit-logging
spec:
  podSelector: {}
  policyTypes:
   - Ingress
   - Egress
  ingress:
   - from:
       - podSelector: {}
  egress:
   - to:
     - namespaceSelector:
        matchLabels:
          kubernetes.io/metadata.name: verify-audit-logging
EOF
```

Example output

```
networkpolicy.networking.k8s.io/deny-all created
networkpolicy.networking.k8s.io/allow-from-same-namespace created
```

2. Create a pod for source traffic in the **default** namespace:

```
$ cat <<EOF| oc create -n default -f -
apiVersion: v1
kind: Pod
metadata:
  name: client
spec:
  containers:
    - name: client
      image: registry.access.redhat.com/rhel7/rhel-tools
      command: ["/bin/sh", "-c"]
      args:
        ["sleep inf"]
EOF
```

3. Create two pods in the **verify-audit-logging** namespace:

```
$ for name in client server; do
```

```
cat <<EOF| oc create -n verify-audit-logging -f -
apiVersion: v1
kind: Pod
metadata:
  name: ${name}
spec:
 containers:
   - name: ${name}
     image: registry.access.redhat.com/rhel7/rhel-tools
     command: ["/bin/sh", "-c"]
     args:
       ["sleep inf"]
EOF
done
```

Example output

```
pod/client created
pod/server created
```

4. To generate traffic and produce network policy audit log entries, complete the following steps:

   a. Obtain the IP address for pod named **server** in the **verify-audit-logging** namespace:

      ```
      $ POD_IP=$(oc get pods server -n verify-audit-logging -o jsonpath='{.status.podIP}')
      ```

   b. Ping the IP address from the previous command from the pod named **client** in the **default** namespace and confirm that all packets are dropped:

      ```
      $ oc exec -it client -n default -- /bin/ping -c 2 $POD_IP
      ```

      Example output

      ```
      PING 10.128.2.55 (10.128.2.55) 56(84) bytes of data.

      --- 10.128.2.55 ping statistics ---
      2 packets transmitted, 0 received, 100% packet loss, time 2041ms
      ```

   c. Ping the IP address saved in the **POD_IP** shell environment variable from the pod named **client** in the **verify-audit-logging** namespace and confirm that all packets are allowed:

      ```
      $ oc exec -it client -n verify-audit-logging -- /bin/ping -c 2 $POD_IP
      ```

      Example output

      ```
      PING 10.128.0.86 (10.128.0.86) 56(84) bytes of data.
      64 bytes from 10.128.0.86: icmp_seq=1 ttl=64 time=2.21 ms
      64 bytes from 10.128.0.86: icmp_seq=2 ttl=64 time=0.440 ms

      --- 10.128.0.86 ping statistics ---
      2 packets transmitted, 2 received, 0% packet loss, time 1001ms
      rtt min/avg/max/mdev = 0.440/1.329/2.219/0.890 ms
      ```

5. Display the latest entries in the network policy audit log:

```
$ for pod in $(oc get pods -n openshift-ovn-kubernetes -l app=ovnkube-node --no-headers=true | awk '{ print $1 }') ; do
    oc exec -it $pod -n openshift-ovn-kubernetes -- tail -4 /var/log/ovn/acl-audit-log.log
  done
```

Example output

```
Defaulting container name to ovn-controller.
Use 'oc describe pod/ovnkube-node-hdb8v -n openshift-ovn-kubernetes' to see all of the
containers in this pod.
2021-06-13T19:33:11.590Z|00005|acl_log(ovn_pinctrl0)|INFO|name="verify-audit-
logging_ingressDefaultDeny", verdict=drop, severity=alert:
icmp,vlan_tci=0x0000,dl_src=0a:58:0a:80:02:39,dl_dst=0a:58:0a:80:02:37,nw_src=10.128.2.57,
nw_dst=10.128.2.55,nw_tos=0,nw_ecn=0,nw_ttl=64,icmp_type=8,icmp_code=0
2021-06-13T19:33:12.614Z|00006|acl_log(ovn_pinctrl0)|INFO|name="verify-audit-
logging_ingressDefaultDeny", verdict=drop, severity=alert:
icmp,vlan_tci=0x0000,dl_src=0a:58:0a:80:02:39,dl_dst=0a:58:0a:80:02:37,nw_src=10.128.2.57,
nw_dst=10.128.2.55,nw_tos=0,nw_ecn=0,nw_ttl=64,icmp_type=8,icmp_code=0
2021-06-13T19:44:10.037Z|00007|acl_log(ovn_pinctrl0)|INFO|name="verify-audit-
logging_allow-from-same-namespace_0", verdict=allow, severity=alert:
icmp,vlan_tci=0x0000,dl_src=0a:58:0a:80:02:3b,dl_dst=0a:58:0a:80:02:3a,nw_src=10.128.2.59,
nw_dst=10.128.2.58,nw_tos=0,nw_ecn=0,nw_ttl=64,icmp_type=8,icmp_code=0
2021-06-13T19:44:11.037Z|00008|acl_log(ovn_pinctrl0)|INFO|name="verify-audit-
logging_allow-from-same-namespace_0", verdict=allow, severity=alert:
icmp,vlan_tci=0x0000,dl_src=0a:58:0a:80:02:3b,dl_dst=0a:58:0a:80:02:3a,nw_src=10.128.2.59,
nw_dst=10.128.2.58,nw_tos=0,nw_ecn=0,nw_ttl=64,icmp_type=8,icmp_code=0
```

27.9.4. Enabling egress firewall and network policy audit logging for a namespace

As a cluster administrator, you can enable audit logging for a namespace.

Prerequisites

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

- Log in to the cluster with a user with **cluster-admin** privileges.

Procedure

- To enable audit logging for a namespace, enter the following command:

```
$ oc annotate namespace <namespace> \
  k8s.ovn.org/acl-logging='{ "deny": "alert", "allow": "notice" }'
```

where:

<namespace>
Specifies the name of the namespace.

TIP

You can alternatively apply the following YAML to enable audit logging:

```
kind: Namespace
apiVersion: v1
metadata:
  name: <namespace>
  annotations:
    k8s.ovn.org/acl-logging: |-
      {
        "deny": "alert",
        "allow": "notice"
      }
```

Example output

```
namespace/verify-audit-logging annotated
```

Verification

- Display the latest entries in the audit log:

  ```
  $ for pod in $(oc get pods -n openshift-ovn-kubernetes -l app=ovnkube-node --no-headers=true | awk '{ print $1 }') ; do
      oc exec -it $pod -n openshift-ovn-kubernetes -- tail -4 /var/log/ovn/acl-audit-log.log
    done
  ```

Example output

```
Defaulting container name to ovn-controller.
Use 'oc describe pod/ovnkube-node-hdb8v -n openshift-ovn-kubernetes' to see all of the
containers in this pod.
2021-06-13T19:33:11.590Z|00005|acl_log(ovn_pinctrl0)|INFO|name="verify-audit-
logging_ingressDefaultDeny", verdict=drop, severity=alert:
icmp,vlan_tci=0x0000,dl_src=0a:58:0a:80:02:39,dl_dst=0a:58:0a:80:02:37,nw_src=10.128.2.57,
nw_dst=10.128.2.55,nw_tos=0,nw_ecn=0,nw_ttl=64,icmp_type=8,icmp_code=0
2021-06-13T19:33:12.614Z|00006|acl_log(ovn_pinctrl0)|INFO|name="verify-audit-
logging_ingressDefaultDeny", verdict=drop, severity=alert:
icmp,vlan_tci=0x0000,dl_src=0a:58:0a:80:02:39,dl_dst=0a:58:0a:80:02:37,nw_src=10.128.2.57,
nw_dst=10.128.2.55,nw_tos=0,nw_ecn=0,nw_ttl=64,icmp_type=8,icmp_code=0
2021-06-13T19:44:10.037Z|00007|acl_log(ovn_pinctrl0)|INFO|name="verify-audit-
logging_allow-from-same-namespace_0", verdict=allow, severity=alert:
icmp,vlan_tci=0x0000,dl_src=0a:58:0a:80:02:3b,dl_dst=0a:58:0a:80:02:3a,nw_src=10.128.2.59,
nw_dst=10.128.2.58,nw_tos=0,nw_ecn=0,nw_ttl=64,icmp_type=8,icmp_code=0
2021-06-13T19:44:11.037Z|00008|acl_log(ovn_pinctrl0)|INFO|name="verify-audit-
logging_allow-from-same-namespace_0", verdict=allow, severity=alert:
icmp,vlan_tci=0x0000,dl_src=0a:58:0a:80:02:3b,dl_dst=0a:58:0a:80:02:3a,nw_src=10.128.2.59,
nw_dst=10.128.2.58,nw_tos=0,nw_ecn=0,nw_ttl=64,icmp_type=8,icmp_code=0
```

27.9.5. Disabling egress firewall and network policy audit logging for a namespace

As a cluster administrator, you can disable audit logging for a namespace.

Prerequisites

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

- Log in to the cluster with a user with **cluster-admin** privileges.

Procedure

- To disable audit logging for a namespace, enter the following command:

      $ oc annotate --overwrite namespace <namespace> k8s.ovn.org/acl-logging-

   where:

   <namespace>
      Specifies the name of the namespace.

   TIP

   You can alternatively apply the following YAML to disable audit logging:

   ```
   kind: Namespace
   apiVersion: v1
   metadata:
     name: <namespace>
     annotations:
       k8s.ovn.org/acl-logging: null
   ```

   Example output

      namespace/verify-audit-logging annotated

27.9.6. Additional resources

- About network policy

- Configuring an egress firewall for a project

27.10. CONFIGURING IPSEC ENCRYPTION

With IPsec enabled, all pod-to-pod network traffic between nodes on the OVN-Kubernetes cluster network is encrypted with IPsec *Transport mode*.

IPsec is disabled by default. It can be enabled either during or after installing the cluster. For information about cluster installation, see OpenShift Container Platform installation overview . If you need to enable IPsec after cluster installation, you must first resize your cluster MTU to account for the overhead of the IPsec ESP IP header.

The following documentation describes how to enable and disable IPSec after cluster installation.

27.10.1. Prerequisites

- You have decreased the size of the cluster MTU by **46** bytes to allow for the additional overhead of the IPsec ESP header. For more information on resizing the MTU that your cluster uses, see Changing the MTU for the cluster network .

27.10.2. Types of network traffic flows encrypted by IPsec

With IPsec enabled, only the following network traffic flows between pods are encrypted:

- Traffic between pods on different nodes on the cluster network

- Traffic from a pod on the host network to a pod on the cluster network

The following traffic flows are not encrypted:

- Traffic between pods on the same node on the cluster network

- Traffic between pods on the host network

- Traffic from a pod on the cluster network to a pod on the host network

The encrypted and unencrypted flows are illustrated in the following diagram:
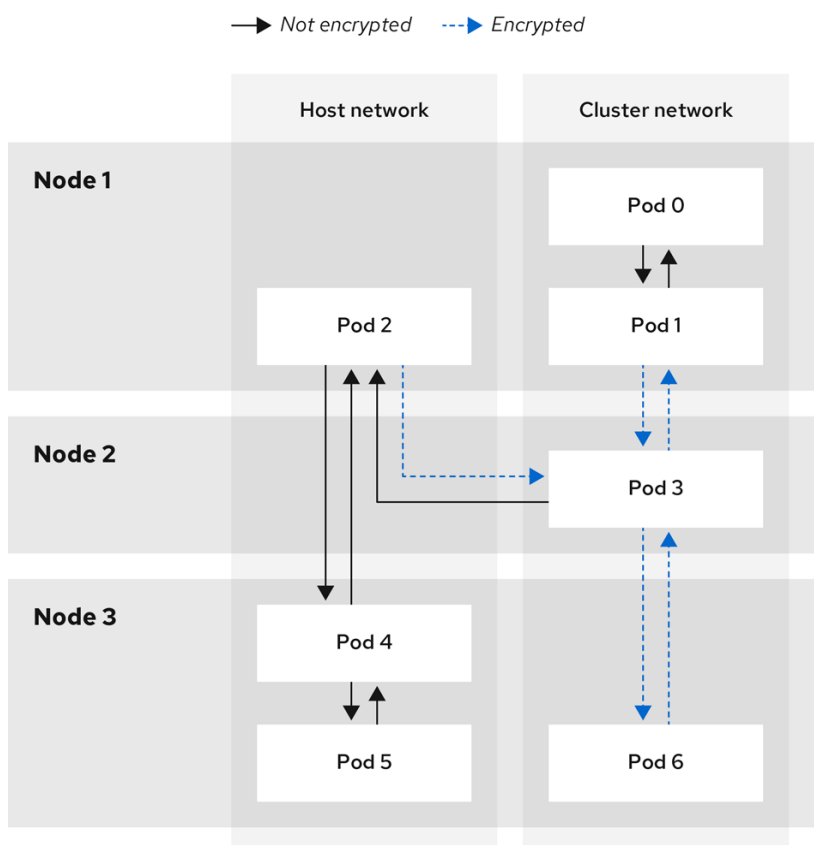
27.10.2.1. Network connectivity requirements when IPsec is enabled

You must configure the network connectivity between machines to allow OpenShift Container Platform cluster components to communicate. Each machine must be able to resolve the hostnames of all other machines in the cluster.

Table 27.12. Ports used for all-machine to all-machine communications

Protocol	Port	Description
UDP	**500**	IPsec IKE packets
	4500	IPsec NAT-T packets
ESP	N/A	IPsec Encapsulating Security Payload (ESP)

27.10.3. Encryption protocol and IPsec mode

The encrypt cipher used is **AES-GCM-16-256**. The integrity check value (ICV) is **16** bytes. The key length is **256** bits.

The IPsec mode used is *Transport mode*, a mode that encrypts end-to-end communication by adding an Encapsulated Security Payload (ESP) header to the IP header of the original packet and encrypts the packet data. OpenShift Container Platform does not currently use or support IPsec *Tunnel mode* for pod-to-pod communication.

27.10.4. Security certificate generation and rotation

The Cluster Network Operator (CNO) generates a self-signed X.509 certificate authority (CA) that is used by IPsec for encryption. Certificate signing requests (CSRs) from each node are automatically fulfilled by the CNO.

The CA is valid for 10 years. The individual node certificates are valid for 5 years and are automatically rotated after 4 1/2 years elapse.

27.10.5. Enabling IPsec encryption

As a cluster administrator, you can enable IPsec encryption after cluster installation.

Prerequisites

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

- Log in to the cluster with a user with **cluster-admin** privileges.

- You have reduced the size of your cluster MTU by **46** bytes to allow for the overhead of the IPsec ESP header.

Procedure

- To enable IPsec encryption, enter the following command:

```
$ oc patch networks.operator.openshift.io cluster --type=merge \
-p '{"spec":{"defaultNetwork":{"ovnKubernetesConfig":{"ipsecConfig":{ }}}}}'
```

Verification

1. To find the names of the OVN-Kubernetes control plane pods, enter the following command:

```
$ oc get pods -l app=ovnkube-master -n openshift-ovn-kubernetes
```

Example output

```
NAME                READY  STATUS   RESTARTS  AGE
ovnkube-master-fvtnh  6/6    Running  0         122m
ovnkube-master-hsgmm  6/6    Running  0         122m
ovnkube-master-qcmdc  6/6    Running  0         122m
```

2. Verify that IPsec is enabled on your cluste by running the following command:

```
$ oc -n openshift-ovn-kubernetes rsh ovnkube-master-<XXXXX> \
  ovn-nbctl --no-leader-only get nb_global . ipsec
```

where:

<XXXXX>

Specifies the random sequence of letters for a pod from the previous step.

Example output

```
true
```

27.10.6. Disabling IPsec encryption

As a cluster administrator, you can disable IPsec encryption only if you enabled IPsec after cluster installation.

> **NOTE**
>
> If you enabled IPsec when you installed your cluster, you cannot disable IPsec with this procedure.

Prerequisites

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

- Log in to the cluster with a user with **cluster-admin** privileges.

Procedure

1. To disable IPsec encryption, enter the following command:

```
$ oc patch networks.operator.openshift.io/cluster --type=json \
  -p='[{"op":"remove", "path":"/spec/defaultNetwork/ovnKubernetesConfig/ipsecConfig"}]'
```

2. Optional: You can increase the size of your cluster MTU by **46** bytes because there is no longer any overhead from the IPsec ESP header in IP packets.

3. Verify that IPsec is disabled on your cluster:

```
$ oc -n openshift-ovn-kubernetes -c nbdb rsh ovnkube-master-<XXXXX> \
  ovn-nbctl --no-leader-only get nb_global . ipsec
```

where:

<XXXXX>

Specifies the random sequence of letters for a pod from the previous step.

Example output

```
false
```

27.10.7. Additional resources

- About the OVN-Kubernetes Container Network Interface (CNI) network plugin

- Changing the MTU for the cluster network

- Network [operator.openshift.io/v1] API

27.11. CONFIGURING AN EGRESS FIREWALL FOR A PROJECT

As a cluster administrator, you can create an egress firewall for a project that restricts egress traffic leaving your OpenShift Container Platform cluster.

27.11.1. How an egress firewall works in a project

As a cluster administrator, you can use an *egress firewall* to limit the external hosts that some or all pods can access from within the cluster. An egress firewall supports the following scenarios:

- A pod can only connect to internal hosts and cannot initiate connections to the public internet.

- A pod can only connect to the public internet and cannot initiate connections to internal hosts that are outside the OpenShift Container Platform cluster.

- A pod cannot reach specified internal subnets or hosts outside the OpenShift Container Platform cluster.

- A pod can connect to only specific external hosts.

For example, you can allow one project access to a specified IP range but deny the same access to a different project. Or you can restrict application developers from updating from Python pip mirrors, and force updates to come only from approved sources.

> **NOTE**
>
> Egress firewall does not apply to the host network namespace. Pods with host networking enabled are unaffected by egress firewall rules.

You configure an egress firewall policy by creating an EgressFirewall custom resource (CR) object. The egress firewall matches network traffic that meets any of the following criteria:

- An IP address range in CIDR format

- A DNS name that resolves to an IP address

- A port number

- A protocol that is one of the following protocols: TCP, UDP, and SCTP

> **IMPORTANT**
>
> If your egress firewall includes a deny rule for **0.0.0.0/0**, access to your OpenShift Container Platform API servers is blocked. You must either add allow rules for each IP address or use the **nodeSelector** type allow rule in your egress policy rules to connect to API servers.
>
> The following example illustrates the order of the egress firewall rules necessary to ensure API server access:
>
> ```
> apiVersion: k8s.ovn.org/v1
> kind: EgressFirewall
> metadata:
>   name: default
>   namespace: <namespace> 1
> spec:
>   egress:
>   - to:
>       cidrSelector: <api_server_address_range> 2
>     type: Allow
> # ...
>   - to:
>       cidrSelector: 0.0.0.0/0 3
>     type: Deny
> ```
>
> 1   The namespace for the egress firewall.
> 2   The IP address range that includes your OpenShift Container Platform API servers.
> 3   A global deny rule prevents access to the OpenShift Container Platform API servers.
>
> To find the IP address for your API servers, run **oc get ep kubernetes -n default**.
>
> For more information, see BZ#1988324.

> **WARNING**
>
> Egress firewall rules do not apply to traffic that goes through routers. Any user with permission to create a Route CR object can bypass egress firewall policy rules by creating a route that points to a forbidden destination.

27.11.1.1. Limitations of an egress firewall

An egress firewall has the following limitations:

- No project can have more than one EgressFirewall object.

- A maximum of one EgressFirewall object with a maximum of 8,000 rules can be defined per project.

- If you are using the OVN-Kubernetes network plugin with shared gateway mode in Red Hat OpenShift Networking, return ingress replies are affected by egress firewall rules. If the egress firewall rules drop the ingress reply destination IP, the traffic is dropped.

Violating any of these restrictions results in a broken egress firewall for the project. Consequently, all external network traffic is dropped, which can cause security risks for your organization.

An Egress Firewall resource can be created in the **kube-node-lease**, **kube-public**, **kube-system**, **openshift** and **openshift-** projects.

27.11.1.2. Matching order for egress firewall policy rules

The egress firewall policy rules are evaluated in the order that they are defined, from first to last. The first rule that matches an egress connection from a pod applies. Any subsequent rules are ignored for that connection.

27.11.1.3. How Domain Name Server (DNS) resolution works

If you use DNS names in any of your egress firewall policy rules, proper resolution of the domain names is subject to the following restrictions:

- Domain name updates are polled based on a time-to-live (TTL) duration. By default, the duration is 30 minutes. When the egress firewall controller queries the local name servers for a domain name, if the response includes a TTL and the TTL is less than 30 minutes, the controller sets the duration for that DNS name to the returned value. Each DNS name is queried after the TTL for the DNS record expires.

- The pod must resolve the domain from the same local name servers when necessary. Otherwise the IP addresses for the domain known by the egress firewall controller and the pod can be different. If the IP addresses for a hostname differ, the egress firewall might not be enforced consistently.

- Because the egress firewall controller and pods asynchronously poll the same local name server, the pod might obtain the updated IP address before the egress controller does, which causes a race condition. Due to this current limitation, domain name usage in EgressFirewall objects is only recommended for domains with infrequent IP address changes.

> **NOTE**
>
> Using DNS names in your egress firewall policy does not affect local DNS resolution through CoreDNS.
>
> However, if your egress firewall policy uses domain names, and an external DNS server handles DNS resolution for an affected pod, you must include egress firewall rules that permit access to the IP addresses of your DNS server.

27.11.2. EgressFirewall custom resource (CR) object

You can define one or more rules for an egress firewall. A rule is either an **Allow** rule or a **Deny** rule, with a specification for the traffic that the rule applies to.

The following YAML describes an EgressFirewall CR object:

EgressFirewall object

```
apiVersion: k8s.ovn.org/v1
kind: EgressFirewall
metadata:
  name: <name> 1
spec:
  egress: 2
    ...
```

1 The name for the object must be **default**.

2 A collection of one or more egress network policy rules as described in the following section.

27.11.2.1. EgressFirewall rules

The following YAML describes an egress firewall rule object. The user can select either an IP address range in CIDR format, a domain name, or use the **nodeSelector** to allow or deny egress traffic. The **egress** stanza expects an array of one or more objects.

Egress policy rule stanza

```
egress:
- type: <type> 1
  to: 2
    cidrSelector: <cidr> 3
    dnsName: <dns_name> 4
    nodeSelector: <label_name>: <label_value> 5
  ports: 6
    ...
```

1 The type of rule. The value must be either **Allow** or **Deny**.

2 A stanza describing an egress traffic match rule that specifies the **cidrSelector** field or the **dnsName** field. You cannot use both fields in the same rule.

3 An IP address range in CIDR format.

4 A DNS domain name.

5 Labels are key/value pairs that the user defines. Labels are attached to objects, such as pods. The **nodeSelector** allows for one or more node labels to be selected and attached to pods.

6 Optional: A stanza describing a collection of network ports and protocols for the rule.

Ports stanza

```
ports:
- port: <port> 1
  protocol: <protocol> 2
```

1 A network port, such as **80** or **443**. If you specify a value for this field, you must also specify a value for **protocol**.

2 A network protocol. The value must be either **TCP**, **UDP**, or **SCTP**.

27.11.2.2. Example EgressFirewall CR objects

The following example defines several egress firewall policy rules:

```
apiVersion: k8s.ovn.org/v1
kind: EgressFirewall
metadata:
  name: default
spec:
  egress: 1
  - type: Allow
    to:
      cidrSelector: 1.2.3.0/24
  - type: Deny
    to:
      cidrSelector: 0.0.0.0/0
```

1 A collection of egress firewall policy rule objects.

The following example defines a policy rule that denies traffic to the host at the **172.16.1.1** IP address, if the traffic is using either the TCP protocol and destination port **80** or any protocol and destination port **443**.

```
apiVersion: k8s.ovn.org/v1
kind: EgressFirewall
metadata:
  name: default
spec:
  egress:
  - type: Deny
    to:
      cidrSelector: 172.16.1.1
    ports:
    - port: 80
      protocol: TCP
    - port: 443
```

27.11.2.3. Example nodeSelector for EgressFirewall

As a cluster administrator, you can allow or deny egress traffic to nodes in your cluster by specifying a label using **nodeSelector**. Labels can be applied to one or more nodes. The following is an example with the **region=east** label:

```
apiVersion: k8s.ovn.org/v1
kind: EgressFirewall
metadata:
  name: default
spec:
  egress:
  - to:
```

```
    nodeSelector:
      matchLabels:
        region: east
  type: Allow
```

TIP

Instead of adding manual rules per node IP address, use node selectors to create a label that allows pods behind an egress firewall to access host network pods.

27.11.3. Creating an egress firewall policy object

As a cluster administrator, you can create an egress firewall policy object for a project.

IMPORTANT

If the project already has an EgressFirewall object defined, you must edit the existing policy to make changes to the egress firewall rules.

Prerequisites

- A cluster that uses the OVN-Kubernetes network plugin.

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

- You must log in to the cluster as a cluster administrator.

Procedure

1. Create a policy rule:

   a. Create a **<policy_name>.yaml** file where **<policy_name>** describes the egress policy rules.

   b. In the file you created, define an egress policy object.

2. Enter the following command to create the policy object. Replace **<policy_name>** with the name of the policy and **<project>** with the project that the rule applies to.

   ```
   $ oc create -f <policy_name>.yaml -n <project>
   ```

   In the following example, a new EgressFirewall object is created in a project named **project1**:

   ```
   $ oc create -f default.yaml -n project1
   ```

   Example output

   ```
   egressfirewall.k8s.ovn.org/v1 created
   ```

3. Optional: Save the **<policy_name>.yaml** file so that you can make changes later.

27.12. VIEWING AN EGRESS FIREWALL FOR A PROJECT

As a cluster administrator, you can list the names of any existing egress firewalls and view the traffic rules for a specific egress firewall.

27.12.1. Viewing an EgressFirewall object

You can view an EgressFirewall object in your cluster.

Prerequisites

- A cluster using the OVN-Kubernetes network plugin.

- Install the OpenShift Command-line Interface (CLI), commonly known as **oc**.

- You must log in to the cluster.

Procedure

1. Optional: To view the names of the EgressFirewall objects defined in your cluster, enter the following command:

   ```
   $ oc get egressfirewall --all-namespaces
   ```

2. To inspect a policy, enter the following command. Replace **<policy_name>** with the name of the policy to inspect.

   ```
   $ oc describe egressfirewall <policy_name>
   ```

   Example output

   ```
   Name:  default
   Namespace: project1
   Created: 20 minutes ago
   Labels:  <none>
   Annotations: <none>
   Rule:  Allow to 1.2.3.0/24
   Rule:  Allow to www.example.com
   Rule:  Deny to 0.0.0.0/0
   ```

27.13. EDITING AN EGRESS FIREWALL FOR A PROJECT

As a cluster administrator, you can modify network traffic rules for an existing egress firewall.

27.13.1. Editing an EgressFirewall object

As a cluster administrator, you can update the egress firewall for a project.

Prerequisites

- A cluster using the OVN-Kubernetes network plugin.

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

- You must log in to the cluster as a cluster administrator.

Procedure

1. Find the name of the EgressFirewall object for the project. Replace **<project>** with the name of the project.

   ```
   $ oc get -n <project> egressfirewall
   ```

2. Optional: If you did not save a copy of the EgressFirewall object when you created the egress network firewall, enter the following command to create a copy.

   ```
   $ oc get -n <project> egressfirewall <name> -o yaml > <filename>.yaml
   ```

   Replace **<project>** with the name of the project. Replace **<name>** with the name of the object. Replace **<filename>** with the name of the file to save the YAML to.

3. After making changes to the policy rules, enter the following command to replace the EgressFirewall object. Replace **<filename>** with the name of the file containing the updated EgressFirewall object.

   ```
   $ oc replace -f <filename>.yaml
   ```

27.14. REMOVING AN EGRESS FIREWALL FROM A PROJECT

As a cluster administrator, you can remove an egress firewall from a project to remove all restrictions on network traffic from the project that leaves the OpenShift Container Platform cluster.

27.14.1. Removing an EgressFirewall object

As a cluster administrator, you can remove an egress firewall from a project.

Prerequisites

- A cluster using the OVN-Kubernetes network plugin.

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

- You must log in to the cluster as a cluster administrator.

Procedure

1. Find the name of the EgressFirewall object for the project. Replace **<project>** with the name of the project.

   ```
   $ oc get -n <project> egressfirewall
   ```

2. Enter the following command to delete the EgressFirewall object. Replace **<project>** with the name of the project and **<name>** with the name of the object.

   ```
   $ oc delete -n <project> egressfirewall <name>
   ```

27.15. CONFIGURING AN EGRESS IP ADDRESS

As a cluster administrator, you can configure the OVN-Kubernetes Container Network Interface (CNI) network plugin to assign one or more egress IP addresses to a namespace, or to specific pods in a namespace.

27.15.1. Egress IP address architectural design and implementation

The OpenShift Container Platform egress IP address functionality allows you to ensure that the traffic from one or more pods in one or more namespaces has a consistent source IP address for services outside the cluster network.

For example, you might have a pod that periodically queries a database that is hosted on a server outside of your cluster. To enforce access requirements for the server, a packet filtering device is configured to allow traffic only from specific IP addresses. To ensure that you can reliably allow access to the server from only that specific pod, you can configure a specific egress IP address for the pod that makes the requests to the server.

An egress IP address assigned to a namespace is different from an egress router, which is used to send traffic to specific destinations.

In some cluster configurations, application pods and ingress router pods run on the same node. If you configure an egress IP address for an application project in this scenario, the IP address is not used when you send a request to a route from the application project.

> **IMPORTANT**
>
> Egress IP addresses must not be configured in any Linux network configuration files, such as **ifcfg-eth0**.

27.15.1.1. Platform support

Support for the egress IP address functionality on various platforms is summarized in the following table:

Platform	Supported
Bare metal	Yes
VMware vSphere	Yes
Red Hat OpenStack Platform (RHOSP)	Yes
Amazon Web Services (AWS)	Yes
Google Cloud Platform (GCP)	Yes
Microsoft Azure	Yes
IBM Z and IBM® LinuxONE	Yes
IBM Z and IBM® LinuxONE for Red Hat Enterprise Linux (RHEL) KVM	Yes

Platform	Supported
IBM Power	Yes

IMPORTANT

The assignment of egress IP addresses to control plane nodes with the EgressIP feature is not supported on a cluster provisioned on Amazon Web Services (AWS). ([BZ#2039656](#))

27.15.1.2. Public cloud platform considerations

For clusters provisioned on public cloud infrastructure, there is a constraint on the absolute number of assignable IP addresses per node. The maximum number of assignable IP addresses per node, or the *IP capacity*, can be described in the following formula:

> IP capacity = public cloud default capacity - sum(current IP assignments)

While the Egress IPs capability manages the IP address capacity per node, it is important to plan for this constraint in your deployments. For example, for a cluster installed on bare-metal infrastructure with 8 nodes you can configure 150 egress IP addresses. However, if a public cloud provider limits IP address capacity to 10 IP addresses per node, the total number of assignable IP addresses is only 80. To achieve the same IP address capacity in this example cloud provider, you would need to allocate 7 additional nodes.

To confirm the IP capacity and subnets for any node in your public cloud environment, you can enter the **oc get node <node_name> -o yaml** command. The **cloud.network.openshift.io/egress-ipconfig** annotation includes capacity and subnet information for the node.

The annotation value is an array with a single object with fields that provide the following information for the primary network interface:

- **interface**: Specifies the interface ID on AWS and Azure and the interface name on GCP.

- **ifaddr**: Specifies the subnet mask for one or both IP address families.

- **capacity**: Specifies the IP address capacity for the node. On AWS, the IP address capacity is provided per IP address family. On Azure and GCP, the IP address capacity includes both IPv4 and IPv6 addresses.

Automatic attachment and detachment of egress IP addresses for traffic between nodes are available. This allows for traffic from many pods in namespaces to have a consistent source IP address to locations outside of the cluster. This also supports OpenShift SDN and OVN-Kubernetes, which is the default networking plugin in Red Hat OpenShift Networking in OpenShift Container Platform 4.13.

NOTE

When an RHOSP cluster administrator assigns a floating IP to the reservation port, OpenShift Container Platform cannot delete the reservation port. The **CloudPrivateIPConfig** object cannot perform delete and move operations until an RHOSP cluster administrator unassigns the floating IP from the reservation port.

The following examples illustrate the annotation from nodes on several public cloud providers. The annotations are indented for readability.

Example cloud.network.openshift.io/egress-ipconfig annotation on AWS

```
cloud.network.openshift.io/egress-ipconfig: [
  {
    "interface":"eni-078d267045138e436",
    "ifaddr":{"ipv4":"10.0.128.0/18"},
    "capacity":{"ipv4":14,"ipv6":15}
  }
]
```

Example cloud.network.openshift.io/egress-ipconfig annotation on GCP

```
cloud.network.openshift.io/egress-ipconfig: [
  {
    "interface":"nic0",
    "ifaddr":{"ipv4":"10.0.128.0/18"},
    "capacity":{"ip":14}
  }
]
```

The following sections describe the IP address capacity for supported public cloud environments for use in your capacity calculation.

27.15.1.2.1. Amazon Web Services (AWS) IP address capacity limits

On AWS, constraints on IP address assignments depend on the instance type configured. For more information, see IP addresses per network interface per instance type

27.15.1.2.2. Google Cloud Platform (GCP) IP address capacity limits

On GCP, the networking model implements additional node IP addresses through IP address aliasing, rather than IP address assignments. However, IP address capacity maps directly to IP aliasing capacity.

The following capacity limits exist for IP aliasing assignment:

- Per node, the maximum number of IP aliases, both IPv4 and IPv6, is 100.

- Per VPC, the maximum number of IP aliases is unspecified, but OpenShift Container Platform scalability testing reveals the maximum to be approximately 15,000.

For more information, see Per instance quotas and Alias IP ranges overview .

27.15.1.2.3. Microsoft Azure IP address capacity limits

On Azure, the following capacity limits exist for IP address assignment:

- Per NIC, the maximum number of assignable IP addresses, for both IPv4 and IPv6, is 256.

- Per virtual network, the maximum number of assigned IP addresses cannot exceed 65,536.

For more information, see Networking limits.

27.15.1.3. Assignment of egress IPs to pods

To assign one or more egress IPs to a namespace or specific pods in a namespace, the following conditions must be satisfied:

- At least one node in your cluster must have the **k8s.ovn.org/egress-assignable: ""** label.

- An **EgressIP** object exists that defines one or more egress IP addresses to use as the source IP address for traffic leaving the cluster from pods in a namespace.

> **IMPORTANT**
>
> If you create **EgressIP** objects prior to labeling any nodes in your cluster for egress IP assignment, OpenShift Container Platform might assign every egress IP address to the first node with the **k8s.ovn.org/egress-assignable: ""** label.
>
> To ensure that egress IP addresses are widely distributed across nodes in the cluster, always apply the label to the nodes you intent to host the egress IP addresses before creating any **EgressIP** objects.

27.15.1.4. Assignment of egress IPs to nodes

When creating an **EgressIP** object, the following conditions apply to nodes that are labeled with the **k8s.ovn.org/egress-assignable: ""** label:

- An egress IP address is never assigned to more than one node at a time.

- An egress IP address is equally balanced between available nodes that can host the egress IP address.

- If the **spec.EgressIPs** array in an **EgressIP** object specifies more than one IP address, the following conditions apply:

  - No node will ever host more than one of the specified IP addresses.

  - Traffic is balanced roughly equally between the specified IP addresses for a given namespace.

- If a node becomes unavailable, any egress IP addresses assigned to it are automatically reassigned, subject to the previously described conditions.

When a pod matches the selector for multiple **EgressIP** objects, there is no guarantee which of the egress IP addresses that are specified in the **EgressIP** objects is assigned as the egress IP address for the pod.

Additionally, if an **EgressIP** object specifies multiple egress IP addresses, there is no guarantee which of the egress IP addresses might be used. For example, if a pod matches a selector for an **EgressIP** object with two egress IP addresses, **10.10.20.1** and **10.10.20.2**, either might be used for each TCP connection or UDP conversation.

27.15.1.5. Architectural diagram of an egress IP address configuration

The following diagram depicts an egress IP address configuration. The diagram describes four pods in two different namespaces running on three nodes in a cluster. The nodes are assigned IP addresses from the **192.168.126.0/18** CIDR block on the host network.

```
Node 1

meta:
  name: node1
  labels:
    k8s.ovn.org/egress-assignable: ""
```

Both Node 1 and Node 3 are labeled with **k8s.ovn.org/egress-assignable: ""** and thus available for the assignment of egress IP addresses.

The dashed lines in the diagram depict the traffic flow from pod1, pod2, and pod3 traveling through the pod network to egress the cluster from Node 1 and Node 3. When an external service receives traffic from any of the pods selected by the example **EgressIP** object, the source IP address is either **192.168.126.10** or **192.168.126.102**. The traffic is balanced roughly equally between these two nodes.

The following resources from the diagram are illustrated in detail:

Namespace objects

The namespaces are defined in the following manifest:

Namespace objects

```
apiVersion: v1
kind: Namespace
metadata:
  name: namespace1
  labels:
    env: prod
---
apiVersion: v1
kind: Namespace
metadata:
  name: namespace2
  labels:
    env: prod
```

EgressIP object

The following **EgressIP** object describes a configuration that selects all pods in any namespace with the **env** label set to **prod**. The egress IP addresses for the selected pods are **192.168.126.10** and **192.168.126.102**.

EgressIP object

```
apiVersion: k8s.ovn.org/v1
kind: EgressIP
metadata:
  name: egressips-prod
spec:
  egressIPs:
  - 192.168.126.10
```

```
    - 192.168.126.102
    namespaceSelector:
      matchLabels:
        env: prod
  status:
    items:
    - node: node1
      egressIP: 192.168.126.10
    - node: node3
      egressIP: 192.168.126.102
```

For the configuration in the previous example, OpenShift Container Platform assigns both egress IP addresses to the available nodes. The **status** field reflects whether and where the egress IP addresses are assigned.

27.15.2. EgressIP object

The following YAML describes the API for the **EgressIP** object. The scope of the object is cluster-wide; it is not created in a namespace.

```
apiVersion: k8s.ovn.org/v1
kind: EgressIP
metadata:
  name: <name>     1
spec:
  egressIPs:       2
  - <ip_address>
  namespaceSelector:   3
    ...
  podSelector:     4
    ...
```

1 The name for the **EgressIPs** object.

2 An array of one or more IP addresses.

3 One or more selectors for the namespaces to associate the egress IP addresses with.

4 Optional: One or more selectors for pods in the specified namespaces to associate egress IP addresses with. Applying these selectors allows for the selection of a subset of pods within a namespace.

The following YAML describes the stanza for the namespace selector:

Namespace selector stanza

```
namespaceSelector:   1
  matchLabels:
    <label_name>: <label_value>
```

1 One or more matching rules for namespaces. If more than one match rule is provided, all matching namespaces are selected.

The following YAML describes the optional stanza for the pod selector:

Pod selector stanza

```
podSelector: 1
  matchLabels:
    <label_name>: <label_value>
```

1 Optional: One or more matching rules for pods in the namespaces that match the specified **namespaceSelector** rules. If specified, only pods that match are selected. Others pods in the namespace are not selected.

In the following example, the **EgressIP** object associates the **192.168.126.11** and **192.168.126.102** egress IP addresses with pods that have the **app** label set to **web** and are in the namespaces that have the **env** label set to **prod**:

Example **EgressIP** object

```
apiVersion: k8s.ovn.org/v1
kind: EgressIP
metadata:
  name: egress-group1
spec:
  egressIPs:
  - 192.168.126.11
  - 192.168.126.102
  podSelector:
    matchLabels:
      app: web
  namespaceSelector:
    matchLabels:
      env: prod
```

In the following example, the **EgressIP** object associates the **192.168.127.30** and **192.168.127.40** egress IP addresses with any pods that do not have the **environment** label set to **development**:

Example **EgressIP** object

```
apiVersion: k8s.ovn.org/v1
kind: EgressIP
metadata:
  name: egress-group2
spec:
  egressIPs:
  - 192.168.127.30
  - 192.168.127.40
  namespaceSelector:
    matchExpressions:
    - key: environment
      operator: NotIn
      values:
      - development
```

27.15.3. EgressIPconfig object

As a feature of egress IP, the **reachabilityTotalTimeoutSeconds** parameter configures the total timeout for checks that are sent by probes to egress IP nodes. The **egressIPConfig** object allows users to set the **reachabilityTotalTimeoutSeconds spec**. If the EgressIP node cannot be reached within this timeout, the node is declared down.

You can increase this value if your network is not stable enough to handle the current default value of 1 second.

The following YAML describes changing the **reachabilityTotalTimeoutSeconds** from the default 1 second probes to 5 second probes:

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  clusterNetwork:
  - cidr: 10.128.0.0/14
    hostPrefix: 23
  defaultNetwork:
    ovnKubernetesConfig:
      egressIPConfig: 1
        reachabilityTotalTimeoutSeconds: 5 2
      gatewayConfig:
        routingViaHost: false
      genevePort: 6081
```

1 The **egressIPConfig** holds the configurations for the options of the **EgressIP** object. Changing these configurations allows you to extend the **EgressIP** object.

2 The value for **reachabilityTotalTimeoutSeconds** accepts integer values from **0** to **60**. A value of 0 disables the reachability check of the egressIP node. Values of **1** to **60** correspond to the duration in seconds between probes sending the reachability check for the node.

27.15.4. Labeling a node to host egress IP addresses

You can apply the **k8s.ovn.org/egress-assignable=""** label to a node in your cluster so that OpenShift Container Platform can assign one or more egress IP addresses to the node.

Prerequisites

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

- Log in to the cluster as a cluster administrator.

Procedure

- To label a node so that it can host one or more egress IP addresses, enter the following command:

```
$ oc label nodes <node_name> k8s.ovn.org/egress-assignable="" 1
```

1 The name of the node to label.

TIP

You can alternatively apply the following YAML to add the label to a node:

```
apiVersion: v1
kind: Node
metadata:
 labels:
   k8s.ovn.org/egress-assignable: ""
 name: <node_name>
```

27.15.5. Next steps

- Assigning egress IPs

27.15.6. Additional resources

- LabelSelector meta/v1

- LabelSelectorRequirement meta/v1

27.16. ASSIGNING AN EGRESS IP ADDRESS

As a cluster administrator, you can assign an egress IP address for traffic leaving the cluster from a namespace or from specific pods in a namespace.

27.16.1. Assigning an egress IP address to a namespace

You can assign one or more egress IP addresses to a namespace or to specific pods in a namespace.

Prerequisites

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

- Log in to the cluster as a cluster administrator.

- Configure at least one node to host an egress IP address.

Procedure

1. Create an **EgressIP** object:

   a. Create a **<egressips_name>.yaml** file where **<egressips_name>** is the name of the object.

   b. In the file that you created, define an **EgressIP** object, as in the following example:

   ```
   apiVersion: k8s.ovn.org/v1
   kind: EgressIP
   metadata:
   ```

```
    name: egress-project1
  spec:
    egressIPs:
    - 192.168.127.10
    - 192.168.127.11
    namespaceSelector:
      matchLabels:
        env: qa
```

2. To create the object, enter the following command.

   ```
   $ oc apply -f <egressips_name>.yaml ❶
   ```

   ❶ Replace **<egressips_name>** with the name of the object.

   Example output

   ```
   egressips.k8s.ovn.org/<egressips_name> created
   ```

3. Optional: Save the **<egressips_name>.yaml** file so that you can make changes later.

4. Add labels to the namespace that requires egress IP addresses. To add a label to the namespace of an **EgressIP** object defined in step 1, run the following command:

   ```
   $ oc label ns <namespace> env=qa ❶
   ```

   ❶ Replace **<namespace>** with the namespace that requires egress IP addresses.

27.16.2. Additional resources

- Configuring egress IP addresses

27.17. CONSIDERATIONS FOR THE USE OF AN EGRESS ROUTER POD

27.17.1. About an egress router pod

The OpenShift Container Platform egress router pod redirects traffic to a specified remote server from a private source IP address that is not used for any other purpose. An egress router pod can send network traffic to servers that are set up to allow access only from specific IP addresses.

> **NOTE**
>
> The egress router pod is not intended for every outgoing connection. Creating large numbers of egress router pods can exceed the limits of your network hardware. For example, creating an egress router pod for every project or application could exceed the number of local MAC addresses that the network interface can handle before reverting to filtering MAC addresses in software.

> **IMPORTANT**
>
> The egress router image is not compatible with Amazon AWS, Azure Cloud, or any other cloud platform that does not support layer 2 manipulations due to their incompatibility with macvlan traffic.

27.17.1.1. Egress router modes

In *redirect mode*, an egress router pod configures **iptables** rules to redirect traffic from its own IP address to one or more destination IP addresses. Client pods that need to use the reserved source IP address must be configured to access the service for the egress router rather than connecting directly to the destination IP. You can access the destination service and port from the application pod by using the **curl** command. For example:

```
$ curl <router_service_IP> <port>
```

> **NOTE**
>
> The egress router CNI plugin supports redirect mode only. This is a difference with the egress router implementation that you can deploy with OpenShift SDN. Unlike the egress router for OpenShift SDN, the egress router CNI plugin does not support HTTP proxy mode or DNS proxy mode.

27.17.1.2. Egress router pod implementation

The egress router implementation uses the egress router Container Network Interface (CNI) plugin. The plugin adds a secondary network interface to a pod.

An egress router is a pod that has two network interfaces. For example, the pod can have **eth0** and **net1** network interfaces. The **eth0** interface is on the cluster network and the pod continues to use the interface for ordinary cluster-related network traffic. The **net1** interface is on a secondary network and has an IP address and gateway for that network. Other pods in the OpenShift Container Platform cluster can access the egress router service and the service enables the pods to access external services. The egress router acts as a bridge between pods and an external system.

Traffic that leaves the egress router exits through a node, but the packets have the MAC address of the **net1** interface from the egress router pod.

When you add an egress router custom resource, the Cluster Network Operator creates the following objects:

- The network attachment definition for the **net1** secondary network interface of the pod.

- A deployment for the egress router.

If you delete an egress router custom resource, the Operator deletes the two objects in the preceding list that are associated with the egress router.

27.17.1.3. Deployment considerations

An egress router pod adds an additional IP address and MAC address to the primary network interface of the node. As a result, you might need to configure your hypervisor or cloud provider to allow the additional address.

Red Hat OpenStack Platform (RHOSP)

If you deploy OpenShift Container Platform on RHOSP, you must allow traffic from the IP and MAC addresses of the egress router pod on your OpenStack environment. If you do not allow the traffic, then communication will fail:

```
$ openstack port set --allowed-address \
  ip_address=<ip_address>,mac_address=<mac_address> <neutron_port_uuid>
```

Red Hat Virtualization (RHV)

If you are using RHV, you must select **No Network Filter** for the Virtual network interface controller (vNIC).

VMware vSphere

If you are using VMware vSphere, see the VMware documentation for securing vSphere standard switches. View and change VMware vSphere default settings by selecting the host virtual switch from the vSphere Web Client.

Specifically, ensure that the following are enabled:

- MAC Address Changes

- Forged Transits

- Promiscuous Mode Operation

27.17.1.4. Failover configuration

To avoid downtime, the Cluster Network Operator deploys the egress router pod as a deployment resource. The deployment name is **egress-router-cni-deployment**. The pod that corresponds to the deployment has a label of **app=egress-router-cni**.

To create a new service for the deployment, use the **oc expose deployment/egress-router-cni-deployment --port <port_number>** command or create a file like the following example:

```
apiVersion: v1
kind: Service
metadata:
  name: app-egress
spec:
  ports:
  - name: tcp-8080
    protocol: TCP
    port: 8080
  - name: tcp-8443
    protocol: TCP
    port: 8443
  - name: udp-80
    protocol: UDP
    port: 80
  type: ClusterIP
  selector:
    app: egress-router-cni
```

27.17.2. Additional resources

- Deploying an egress router in redirection mode

27.18. DEPLOYING AN EGRESS ROUTER POD IN REDIRECT MODE

As a cluster administrator, you can deploy an egress router pod to redirect traffic to specified destination IP addresses from a reserved source IP address.

The egress router implementation uses the egress router Container Network Interface (CNI) plugin.

27.18.1. Egress router custom resource

Define the configuration for an egress router pod in an egress router custom resource. The following YAML describes the fields for the configuration of an egress router in redirect mode:

```
apiVersion: network.operator.openshift.io/v1
kind: EgressRouter
metadata:
  name: <egress_router_name>
  namespace: <namespace>  <.>
spec:
  addresses: [  <.>
    {
      ip: "<egress_router>",  <.>
      gateway: "<egress_gateway>"  <.>
    }
  ]
  mode: Redirect
  redirect: {
    redirectRules: [  <.>
      {
        destinationIP: "<egress_destination>",
        port: <egress_router_port>,
        targetPort: <target_port>,  <.>
        protocol: <network_protocol>  <.>
      },
      ...
    ],
    fallbackIP: "<egress_destination>" <.>
  }
```

<.> Optional: The **namespace** field specifies the namespace to create the egress router in. If you do not specify a value in the file or on the command line, the **default** namespace is used.

<.> The **addresses** field specifies the IP addresses to configure on the secondary network interface.

<.> The **ip** field specifies the reserved source IP address and netmask from the physical network that the node is on to use with egress router pod. Use CIDR notation to specify the IP address and netmask.

<.> The **gateway** field specifies the IP address of the network gateway.

<.> Optional: The **redirectRules** field specifies a combination of egress destination IP address, egress router port, and protocol. Incoming connections to the egress router on the specified port and protocol are routed to the destination IP address.

<.> Optional: The **targetPort** field specifies the network port on the destination IP address. If this field is not specified, traffic is routed to the same network port that it arrived on.

<.> The **protocol** field supports TCP, UDP, or SCTP.

<.> Optional: The **fallbackIP** field specifies a destination IP address. If you do not specify any redirect rules, the egress router sends all traffic to this fallback IP address. If you specify redirect rules, any connections to network ports that are not defined in the rules are sent by the egress router to this fallback IP address. If you do not specify this field, the egress router rejects connections to network ports that are not defined in the rules.

Example egress router specification

```
apiVersion: network.operator.openshift.io/v1
kind: EgressRouter
metadata:
  name: egress-router-redirect
spec:
  networkInterface: {
    macvlan: {
      mode: "Bridge"
    }
  }
  addresses: [
    {
      ip: "192.168.12.99/24",
      gateway: "192.168.12.1"
    }
  ]
  mode: Redirect
  redirect: {
    redirectRules: [
      {
        destinationIP: "10.0.0.99",
        port: 80,
        protocol: UDP
      },
      {
        destinationIP: "203.0.113.26",
        port: 8080,
        targetPort: 80,
        protocol: TCP
      },
      {
        destinationIP: "203.0.113.27",
        port: 8443,
        targetPort: 443,
        protocol: TCP
      }
    ]
  }
```

27.18.2. Deploying an egress router in redirect mode

You can deploy an egress router to redirect traffic from its own reserved source IP address to one or more destination IP addresses.

After you add an egress router, the client pods that need to use the reserved source IP address must be modified to connect to the egress router rather than connecting directly to the destination IP.

Prerequisites

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

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

Procedure

1. Create an egress router definition.

2. To ensure that other pods can find the IP address of the egress router pod, create a service that uses the egress router, as in the following example:

```
apiVersion: v1
kind: Service
metadata:
  name: egress-1
spec:
  ports:
  - name: web-app
    protocol: TCP
    port: 8080
  type: ClusterIP
  selector:
    app: egress-router-cni <.>
```

<.> Specify the label for the egress router. The value shown is added by the Cluster Network Operator and is not configurable.

After you create the service, your pods can connect to the service. The egress router pod redirects traffic to the corresponding port on the destination IP address. The connections originate from the reserved source IP address.

Verification

To verify that the Cluster Network Operator started the egress router, complete the following procedure:

1. View the network attachment definition that the Operator created for the egress router:

```
$ oc get network-attachment-definition egress-router-cni-nad
```

The name of the network attachment definition is not configurable.

Example output

```
NAME                   AGE
egress-router-cni-nad   18m
```

2. View the deployment for the egress router pod:

```
$ oc get deployment egress-router-cni-deployment
```

The name of the deployment is not configurable.

Example output

```
NAME                       READY  UP-TO-DATE  AVAILABLE  AGE
egress-router-cni-deployment  1/1    1           1          18m
```

3. View the status of the egress router pod:

```
$ oc get pods -l app=egress-router-cni
```

Example output

```
NAME                                READY  STATUS   RESTARTS  AGE
egress-router-cni-deployment-575465c75c-qkq6m  1/1    Running  0         18m
```

4. View the logs and the routing table for the egress router pod.

a. Get the node name for the egress router pod:

```
$ POD_NODENAME=$(oc get pod -l app=egress-router-cni -o jsonpath="
{.items[0].spec.nodeName}")
```

b. Enter into a debug session on the target node. This step instantiates a debug pod called **<node_name>-debug**:

```
$ oc debug node/$POD_NODENAME
```

c. Set /**host** as the root directory within the debug shell. The debug pod mounts the root file system of the host in /**host** within the pod. By changing the root directory to /**host**, you can run binaries from the executable paths of the host:

```
# chroot /host
```

d. From within the **chroot** environment console, display the egress router logs:

```
# cat /tmp/egress-router-log
```

Example output

```
2021-04-26T12:27:20Z [debug] Called CNI ADD
2021-04-26T12:27:20Z [debug] Gateway: 192.168.12.1
2021-04-26T12:27:20Z [debug] IP Source Addresses: [192.168.12.99/24]
2021-04-26T12:27:20Z [debug] IP Destinations: [80 UDP 10.0.0.99/30 8080 TCP
203.0.113.26/30 80 8443 TCP 203.0.113.27/30 443]
2021-04-26T12:27:20Z [debug] Created macvlan interface
2021-04-26T12:27:20Z [debug] Renamed macvlan to "net1"
2021-04-26T12:27:20Z [debug] Adding route to gateway 192.168.12.1 on macvlan interface
```

2021-04-26T12:27:20Z [debug] deleted default route {Ifindex: 3 Dst: <nil> Src: <nil> Gw: 10.128.10.1 Flags: [] Table: 254}
2021-04-26T12:27:20Z [debug] Added new default route with gateway 192.168.12.1
2021-04-26T12:27:20Z [debug] Added iptables rule: iptables -t nat PREROUTING -i eth0 -p UDP --dport 80 -j DNAT --to-destination 10.0.0.99
2021-04-26T12:27:20Z [debug] Added iptables rule: iptables -t nat PREROUTING -i eth0 -p TCP --dport 8080 -j DNAT --to-destination 203.0.113.26:80
2021-04-26T12:27:20Z [debug] Added iptables rule: iptables -t nat PREROUTING -i eth0 -p TCP --dport 8443 -j DNAT --to-destination 203.0.113.27:443
2021-04-26T12:27:20Z [debug] Added iptables rule: iptables -t nat -o net1 -j SNAT --to-source 192.168.12.99

The logging file location and logging level are not configurable when you start the egress router by creating an **EgressRouter** object as described in this procedure.

e. From within the **chroot** environment console, get the container ID:

```
# crictl ps --name egress-router-cni-pod | awk '{print $1}'
```

Example output

```
CONTAINER
bac9fae69ddb6
```

f. Determine the process ID of the container. In this example, the container ID is **bac9fae69ddb6**:

```
# crictl inspect -o yaml bac9fae69ddb6 | grep 'pid:' | awk '{print $2}'
```

Example output

```
68857
```

g. Enter the network namespace of the container:

```
# nsenter -n -t 68857
```

h. Display the routing table:

```
# ip route
```

In the following example output, the **net1** network interface is the default route. Traffic for the cluster network uses the **eth0** network interface. Traffic for the **192.168.12.0/24** network uses the **net1** network interface and originates from the reserved source IP address **192.168.12.99**. The pod routes all other traffic to the gateway at IP address **192.168.12.1**. Routing for the service network is not shown.

Example output

```
default via 192.168.12.1 dev net1
10.128.10.0/23 dev eth0 proto kernel scope link src 10.128.10.18
192.168.12.0/24 dev net1 proto kernel scope link src 192.168.12.99
192.168.12.1 dev net1
```

27.19. ENABLING MULTICAST FOR A PROJECT

27.19.1. About multicast

With IP multicast, data is broadcast to many IP addresses simultaneously.

> **IMPORTANT**
>
> - At this time, multicast is best used for low-bandwidth coordination or service discovery and not a high-bandwidth solution.
>
> - By default, network policies affect all connections in a namespace. However, multicast is unaffected by network policies. If multicast is enabled in the same namespace as your network policies, it is always allowed, even if there is a **deny-all** network policy. Cluster administrators should consider the implications to the exemption of multicast from network policies before enabling it.

Multicast traffic between OpenShift Container Platform pods is disabled by default. If you are using the OVN-Kubernetes network plugin, you can enable multicast on a per-project basis.

27.19.2. Enabling multicast between pods

You can enable multicast between pods for your project.

Prerequisites

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

- You must log in to the cluster with a user that has the **cluster-admin** role.

Procedure

- Run the following command to enable multicast for a project. Replace **<namespace>** with the namespace for the project you want to enable multicast for.

  ```
  $ oc annotate namespace <namespace> \
      k8s.ovn.org/multicast-enabled=true
  ```

  TIP

  You can alternatively apply the following YAML to add the annotation:

  ```
  apiVersion: v1
  kind: Namespace
  metadata:
    name: <namespace>
    annotations:
      k8s.ovn.org/multicast-enabled: "true"
  ```

Verification

To verify that multicast is enabled for a project, complete the following procedure:

1. Change your current project to the project that you enabled multicast for. Replace **<project>** with the project name.

   ```
   $ oc project <project>
   ```

2. Create a pod to act as a multicast receiver:

   ```
   $ cat <<EOF| oc create -f -
   apiVersion: v1
   kind: Pod
   metadata:
     name: mlistener
     labels:
       app: multicast-verify
   spec:
     containers:
       - name: mlistener
         image: registry.access.redhat.com/ubi9
         command: ["/bin/sh", "-c"]
         args:
           ["dnf -y install socat hostname && sleep inf"]
         ports:
           - containerPort: 30102
             name: mlistener
             protocol: UDP
   EOF
   ```

3. Create a pod to act as a multicast sender:

   ```
   $ cat <<EOF| oc create -f -
   apiVersion: v1
   kind: Pod
   metadata:
     name: msender
     labels:
       app: multicast-verify
   spec:
     containers:
       - name: msender
         image: registry.access.redhat.com/ubi9
         command: ["/bin/sh", "-c"]
         args:
           ["dnf -y install socat && sleep inf"]
   EOF
   ```

4. In a new terminal window or tab, start the multicast listener.

   a. Get the IP address for the Pod:

      ```
      $ POD_IP=$(oc get pods mlistener -o jsonpath='{.status.podIP}')
      ```

   b. Start the multicast listener by entering the following command:

```
$ oc exec mlistener -i -t -- \
    socat UDP4-RECVFROM:30102,ip-add-membership=224.1.0.1:$POD_IP,fork
EXEC:hostname
```

5. Start the multicast transmitter.

    a. Get the pod network IP address range:

    ```
    $ CIDR=$(oc get Network.config.openshift.io cluster \
        -o jsonpath='{.status.clusterNetwork[0].cidr}')
    ```

    b. To send a multicast message, enter the following command:

    ```
    $ oc exec msender -i -t -- \
        /bin/bash -c "echo | socat STDIO UDP4-
    DATAGRAM:224.1.0.1:30102,range=$CIDR,ip-multicast-ttl=64"
    ```

    If multicast is working, the previous command returns the following output:

    ```
    mlistener
    ```

27.20. DISABLING MULTICAST FOR A PROJECT

27.20.1. Disabling multicast between pods

You can disable multicast between pods for your project.

Prerequisites

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

- You must log in to the cluster with a user that has the **cluster-admin** role.

Procedure

- Disable multicast by running the following command:

    ```
    $ oc annotate namespace <namespace> \ 1
        k8s.ovn.org/multicast-enabled-
    ```

    1   The **namespace** for the project you want to disable multicast for.

TIP

You can alternatively apply the following YAML to delete the annotation:

```
apiVersion: v1
kind: Namespace
metadata:
  name: <namespace>
  annotations:
    k8s.ovn.org/multicast-enabled: null
```

27.21. TRACKING NETWORK FLOWS

As a cluster administrator, you can collect information about pod network flows from your cluster to assist with the following areas:

- Monitor ingress and egress traffic on the pod network.

- Troubleshoot performance issues.

- Gather data for capacity planning and security audits.

When you enable the collection of the network flows, only the metadata about the traffic is collected. For example, packet data is not collected, but the protocol, source address, destination address, port numbers, number of bytes, and other packet-level information is collected.

The data is collected in one or more of the following record formats:

- NetFlow

- sFlow

- IPFIX

When you configure the Cluster Network Operator (CNO) with one or more collector IP addresses and port numbers, the Operator configures Open vSwitch (OVS) on each node to send the network flows records to each collector.

You can configure the Operator to send records to more than one type of network flow collector. For example, you can send records to NetFlow collectors and also send records to sFlow collectors.

When OVS sends data to the collectors, each type of collector receives identical records. For example, if you configure two NetFlow collectors, OVS on a node sends identical records to the two collectors. If you also configure two sFlow collectors, the two sFlow collectors receive identical records. However, each collector type has a unique record format.

Collecting the network flows data and sending the records to collectors affects performance. Nodes process packets at a slower rate. If the performance impact is too great, you can delete the destinations for collectors to disable collecting network flows data and restore performance.

NOTE

Enabling network flow collectors might have an impact on the overall performance of the cluster network.

27.21.1. Network object configuration for tracking network flows

The fields for configuring network flows collectors in the Cluster Network Operator (CNO) are shown in the following table:

Table 27.13. Network flows configuration

Field	Type	Description
metadata.name	**string**	The name of the CNO object. This name is always **cluster**.
spec.exportNet workFlows	**object**	One or more of **netFlow**, **sFlow**, or **ipfix**.
spec.exportNet workFlows.netF low.collectors	**array**	A list of IP address and network port pairs for up to 10 collectors.
spec.exportNet workFlows.sFlo w.collectors	**array**	A list of IP address and network port pairs for up to 10 collectors.
spec.exportNet workFlows.ipfix. collectors	**array**	A list of IP address and network port pairs for up to 10 collectors.

After applying the following manifest to the CNO, the Operator configures Open vSwitch (OVS) on each node in the cluster to send network flows records to the NetFlow collector that is listening at **192.168.1.99:2056**.

Example configuration for tracking network flows

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  exportNetworkFlows:
    netFlow:
      collectors:
        - 192.168.1.99:2056
```

27.21.2. Adding destinations for network flows collectors

As a cluster administrator, you can configure the Cluster Network Operator (CNO) to send network flows metadata about the pod network to a network flows collector.

Prerequisites

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

- You are logged in to the cluster with a user with **cluster-admin** privileges.

- You have a network flows collector and know the IP address and port that it listens on.

Procedure

1. Create a patch file that specifies the network flows collector type and the IP address and port information of the collectors:

   ```
   spec:
     exportNetworkFlows:
       netFlow:
         collectors:
           - 192.168.1.99:2056
   ```

2. Configure the CNO with the network flows collectors:

   ```
   $ oc patch network.operator cluster --type merge -p "$(cat <file_name>.yaml)"
   ```

   Example output

   ```
   network.operator.openshift.io/cluster patched
   ```

Verification

Verification is not typically necessary. You can run the following command to confirm that Open vSwitch (OVS) on each node is configured to send network flows records to one or more collectors.

1. View the Operator configuration to confirm that the **exportNetworkFlows** field is configured:

   ```
   $ oc get network.operator cluster -o jsonpath="{.spec.exportNetworkFlows}"
   ```

   Example output

   ```
   {"netFlow":{"collectors":["192.168.1.99:2056"]}}
   ```

2. View the network flows configuration in OVS from each node:

   ```
   $ for pod in $(oc get pods -n openshift-ovn-kubernetes -l app=ovnkube-node -o
   jsonpath='{range@.items[*]}{.metadata.name}{"\n"}{end}');
     do ;
       echo;
       echo $pod;
       oc -n openshift-ovn-kubernetes exec -c ovnkube-node $pod \
         -- bash -c 'for type in ipfix sflow netflow ; do ovs-vsctl find $type ; done';
   done
   ```

   Example output

   ```
   ovnkube-node-xrn4p
   _uuid               : a4d2aaca-5023-4f3d-9400-7275f92611f9
   active_timeout      : 60
   add_id_to_interface : false
   engine_id           : []
   engine_type         : []
   ```

```
external_ids     : {}
targets          : ["192.168.1.99:2056"]

ovnkube-node-z4vq9
_uuid            : 61d02fdb-9228-4993-8ff5-b27f01a29bd6
active_timeout     : 60
add_id_to_interface : false
engine_id        : []
engine_type      : []
external_ids     : {}
targets          : ["192.168.1.99:2056"]-


...
```

27.21.3. Deleting all destinations for network flows collectors

As a cluster administrator, you can configure the Cluster Network Operator (CNO) to stop sending network flows metadata to a network flows collector.

Prerequisites

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

- You are logged in to the cluster with a user with **cluster-admin** privileges.

Procedure

1. Remove all network flows collectors:

   ```
   $ oc patch network.operator cluster --type='json' \
       -p='[{"op":"remove", "path":"/spec/exportNetworkFlows"}]'
   ```

   Example output

   ```
   network.operator.openshift.io/cluster patched
   ```

27.21.4. Additional resources

- Network [operator.openshift.io/v1]

27.22. CONFIGURING HYBRID NETWORKING

As a cluster administrator, you can configure the Red Hat OpenShift Networking OVN-Kubernetes network plugin to allow Linux and Windows nodes to host Linux and Windows workloads, respectively.

27.22.1. Configuring hybrid networking with OVN-Kubernetes

You can configure your cluster to use hybrid networking with the OVN-Kubernetes network plugin. This allows a hybrid cluster that supports different node networking configurations.

> **NOTE**
>
> This configuration is necessary to run both Linux and Windows nodes in the same cluster.

Prerequisites

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

- Log in to the cluster with a user with **cluster-admin** privileges.

- Ensure that the cluster uses the OVN-Kubernetes network plugin.

Procedure

1. To configure the OVN-Kubernetes hybrid network overlay, enter the following command:

```
$ oc patch networks.operator.openshift.io cluster --type=merge \
 -p '{
  "spec":{
   "defaultNetwork":{
    "ovnKubernetesConfig":{
     "hybridOverlayConfig":{
      "hybridClusterNetwork":[
       {
         "cidr": "<cidr>",
         "hostPrefix": <prefix>
       }
      ],
      "hybridOverlayVXLANPort": <overlay_port>
     }
    }
   }
  }
 }'
```

where:

cidr

Specify the CIDR configuration used for nodes on the additional overlay network. This CIDR cannot overlap with the cluster network CIDR.

hostPrefix

Specifies the subnet prefix length to assign to each individual node. For example, if **hostPrefix** is set to **23**, then each node is assigned a /**23** subnet out of the given **cidr**, which allows for 510 (2^(32 - 23) - 2) pod IP addresses. If you are required to provide access to nodes from an external network, configure load balancers and routers to manage the traffic.

hybridOverlayVXLANPort

Specify a custom VXLAN port for the additional overlay network. This is required for running Windows nodes in a cluster installed on vSphere, and must not be configured for any other cloud provider. The custom port can be any open port excluding the default **4789** port. For more information on this requirement, see the Microsoft documentation on Pod-to-pod connectivity between hosts is broken.
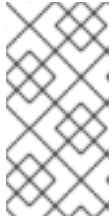
NOTE

Windows Server Long-Term Servicing Channel (LTSC): Windows Server 2019 is not supported on clusters with a custom **hybridOverlayVXLANPort** value because this Windows server version does not support selecting a custom VXLAN port.

Example output

```
network.operator.openshift.io/cluster patched
```

2. To confirm that the configuration is active, enter the following command. It can take several minutes for the update to apply.

```
$ oc get network.operator.openshift.io -o jsonpath="{.items[0].spec.defaultNetwork.ovnKubernetesConfig}"
```

27.22.2. Additional resources

- [Understanding Windows container workloads](#)

- [Enabling Windows container workloads](#)

- [Installing a cluster on AWS with network customizations](#)

- [Installing a cluster on Azure with network customizations](#)

CHAPTER 28. OPENSHIFT SDN NETWORK PLUGIN

28.1. ABOUT THE OPENSHIFT SDN NETWORK PLUGIN

Part of Red Hat OpenShift Networking, OpenShift SDN is a network plugin that uses a software-defined networking (SDN) approach to provide a unified cluster network that enables communication between pods across the OpenShift Container Platform cluster. This pod network is established and maintained by OpenShift SDN, which configures an overlay network using Open vSwitch (OVS).

28.1.1. OpenShift SDN network isolation modes

OpenShift SDN provides three SDN modes for configuring the pod network:

- *Network policy* mode allows project administrators to configure their own isolation policies using **NetworkPolicy** objects. Network policy is the default mode in OpenShift Container Platform 4.13.

- *Multitenant* mode provides project-level isolation for pods and services. Pods from different projects cannot send packets to or receive packets from pods and services of a different project. You can disable isolation for a project, allowing it to send network traffic to all pods and services in the entire cluster and receive network traffic from those pods and services.

- *Subnet* mode provides a flat pod network where every pod can communicate with every other pod and service. The network policy mode provides the same functionality as subnet mode.

28.1.2. Supported network plugin feature matrix

Red Hat OpenShift Networking offers two options for the network plugin, OpenShift SDN and OVN-Kubernetes, for the network plugin. The following table summarizes the current feature support for both network plugins:

Table 28.1. Default CNI network plugin feature comparison

Feature	OpenShift SDN	OVN-Kubernetes
Egress IPs	Supported	Supported
Egress firewall [1]	Supported	Supported
Egress router	Supported	Supported [2]
Hybrid networking	Not supported	Supported
IPsec encryption	Not supported	Supported
IPv6	Not supported	Supported [3] [4]
Kubernetes network policy	Supported	Supported
Kubernetes network policy logs	Not supported	Supported

Feature	OpenShift SDN	OVN-Kubernetes
Multicast	Supported	Supported
Hardware offloading	Not supported	Supported

1. Egress firewall is also known as egress network policy in OpenShift SDN. This is not the same as network policy egress.

2. Egress router for OVN-Kubernetes supports only redirect mode.

3. IPv6 is supported only on bare metal, IBM Power, and IBM Z clusters.

4. IPv6 single stack does not support Kubernetes NMState and is not supported on IBM Power and IBM Z clusters.

28.2. MIGRATING TO THE OPENSHIFT SDN NETWORK PLUGIN

As a cluster administrator, you can migrate to the OpenShift SDN network plugin from the OVN-Kubernetes network plugin.

To learn more about OpenShift SDN, read About the OpenShift SDN network plugin .

28.2.1. How the migration process works

The following table summarizes the migration process by segmenting between the user-initiated steps in the process and the actions that the migration performs in response.

Table 28.2. Migrating to OpenShift SDN from OVN-Kubernetes

User-initiated steps	Migration activity
Set the **migration** field of the **Network.operator.openshift.io** custom resource (CR) named **cluster** to **OpenShiftSDN**. Make sure the **migration** field is **null** before setting it to a value.	**Cluster Network Operator (CNO)** Updates the status of the **Network.config.openshift.io** CR named **cluster** accordingly. **Machine Config Operator (MCO)** Rolls out an update to the systemd configuration necessary for OpenShift SDN; the MCO updates a single machine per pool at a time by default, causing the total time the migration takes to increase with the size of the cluster.

User-initiated steps	Migration activity
Update the **networkType** field of the **Network.config.openshift.io** CR.	**CNO** Performs the following actions: • Destroys the OVN-Kubernetes control plane pods. • Deploys the OpenShift SDN control plane pods. • Updates the Multus objects to reflect the new network plugin.
Reboot each node in the cluster.	**Cluster** As nodes reboot, the cluster assigns IP addresses to pods on the OpenShift SDN cluster network.

28.2.2. Migrating to the OpenShift SDN network plugin

As a cluster administrator, you can migrate to the OpenShift SDN Container Network Interface (CNI) network plugin. During the migration you must reboot every node in your cluster.

Prerequisites

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

- Access to the cluster as a user with the **cluster-admin** role.

- A cluster installed on infrastructure configured with the OVN-Kubernetes network plugin.

- A recent backup of the etcd database is available.

- A reboot can be triggered manually for each node.

- The cluster is in a known good state, without any errors.

Procedure

1. Stop all of the machine configuration pools managed by the Machine Config Operator (MCO):

   - Stop the master configuration pool:

     ```
     $ oc patch MachineConfigPool master --type='merge' --patch \
       '{ "spec": { "paused": true } }'
     ```

   - Stop the worker machine configuration pool:

     ```
     $ oc patch MachineConfigPool worker --type='merge' --patch \
       '{ "spec":{ "paused": true } }'
     ```

–

2. To prepare for the migration, set the migration field to **null** by entering the following command:

```
$ oc patch Network.operator.openshift.io cluster --type='merge' \
  --patch '{ "spec": { "migration": null } }'
```

3. To start the migration, set the network plugin back to OpenShift SDN by entering the following commands:

```
$ oc patch Network.operator.openshift.io cluster --type='merge' \
  --patch '{ "spec": { "migration": { "networkType": "OpenShiftSDN" } } }'
```

```
$ oc patch Network.config.openshift.io cluster --type='merge' \
  --patch '{ "spec": { "networkType": "OpenShiftSDN" } }'
```

4. Optional: You can disable automatic migration of several OVN-Kubernetes capabilities to the OpenShift SDN equivalents:

   - Egress IPs

   - Egress firewall

   - Multicast

   To disable automatic migration of the configuration for any of the previously noted OpenShift SDN features, specify the following keys:

```
$ oc patch Network.operator.openshift.io cluster --type='merge' \
  --patch '{
    "spec": {
      "migration": {
        "networkType": "OpenShiftSDN",
        "features": {
          "egressIP": <bool>,
          "egressFirewall": <bool>,
          "multicast": <bool>
        }
      }
    }
  }'
```

   where:

   bool: Specifies whether to enable migration of the feature. The default is **true**.

5. Optional: You can customize the following settings for OpenShift SDN to meet your network infrastructure requirements:

   - Maximum transmission unit (MTU)

   - VXLAN port

   To customize either or both of the previously noted settings, customize and enter the following command. If you do not need to change the default value, omit the key from the patch.

```
$ oc patch Network.operator.openshift.io cluster --type=merge \
```

```
  --patch '{
    "spec":{
     "defaultNetwork":{
      "openshiftSDNConfig":{
        "mtu":<mtu>,
        "vxlanPort":<port>
    }}}}'
```

mtu

> The MTU for the VXLAN overlay network. This value is normally configured automatically, but if the nodes in your cluster do not all use the same MTU, then you must set this explicitly to **50** less than the smallest node MTU value.

port

> The UDP port for the VXLAN overlay network. If a value is not specified, the default is **4789**. The port cannot be the same as the Geneve port that is used by OVN-Kubernetes. The default value for the Geneve port is **6081**.

Example patch command

```
$ oc patch Network.operator.openshift.io cluster --type=merge \
  --patch '{
    "spec":{
     "defaultNetwork":{
      "openshiftSDNConfig":{
        "mtu":1200
    }}}}'
```

6. Reboot each node in your cluster. You can reboot the nodes in your cluster with either of the following approaches:

   - With the **oc rsh** command, you can use a bash script similar to the following:

     ```
     #!/bin/bash
     readarray -t POD_NODES <<< "$(oc get pod -n openshift-machine-config-operator -o wide| grep daemon|awk '{print $1" "$7}')"

     for i in "${POD_NODES[@]}"
     do
       read -r POD NODE <<< "$i"
       until oc rsh -n openshift-machine-config-operator "$POD" chroot /rootfs shutdown -r +1
         do
           echo "cannot reboot node $NODE, retry" && sleep 3
         done
     done
     ```

   - With the **ssh** command, you can use a bash script similar to the following. The script assumes that you have configured sudo to not prompt for a password.

     ```
     #!/bin/bash

     for ip in $(oc get nodes  -o jsonpath='{.items[*].status.addresses[?
     (@.type=="InternalIP")].address}')
     do
     ```

```
     echo "reboot node $ip"
     ssh -o StrictHostKeyChecking=no core@$ip sudo shutdown -r -t 3
done
```

7. Wait until the Multus daemon set rollout completes. Run the following command to see your rollout status:

```
$ oc -n openshift-multus rollout status daemonset/multus
```

The name of the Multus pods is in the form of **multus-<xxxxx>** where **<xxxxx>** is a random sequence of letters. It might take several moments for the pods to restart.

Example output

```
Waiting for daemon set "multus" rollout to finish: 1 out of 6 new pods have been updated...
...
Waiting for daemon set "multus" rollout to finish: 5 of 6 updated pods are available...
daemon set "multus" successfully rolled out
```

8. After the nodes in your cluster have rebooted and the multus pods are rolled out, start all of the machine configuration pools by running the following commands::

   - Start the master configuration pool:

     ```
     $ oc patch MachineConfigPool master --type='merge' --patch \
       '{ "spec": { "paused": false } }'
     ```

   - Start the worker configuration pool:

     ```
     $ oc patch MachineConfigPool worker --type='merge' --patch \
       '{ "spec": { "paused": false } }'
     ```

   As the MCO updates machines in each config pool, it reboots each node.

   By default the MCO updates a single machine per pool at a time, so the time that the migration requires to complete grows with the size of the cluster.

9. Confirm the status of the new machine configuration on the hosts:

   a. To list the machine configuration state and the name of the applied machine configuration, enter the following command:

      ```
      $ oc describe node | egrep "hostname|machineconfig"
      ```

      Example output

      ```
      kubernetes.io/hostname=master-0
      machineconfiguration.openshift.io/currentConfig: rendered-master-
      c53e221d9d24e1c8bb6ee89dd3d8ad7b
      machineconfiguration.openshift.io/desiredConfig: rendered-master-
      c53e221d9d24e1c8bb6ee89dd3d8ad7b
      machineconfiguration.openshift.io/reason:
      machineconfiguration.openshift.io/state: Done
      ```

Verify that the following statements are true:

- The value of **machineconfiguration.openshift.io/state** field is **Done**.

- The value of the **machineconfiguration.openshift.io/currentConfig** field is equal to the value of the **machineconfiguration.openshift.io/desiredConfig** field.

b. To confirm that the machine config is correct, enter the following command:

```
$ oc get machineconfig <config_name> -o yaml
```

where **<config_name>** is the name of the machine config from the **machineconfiguration.openshift.io/currentConfig** field.

10. Confirm that the migration succeeded:

a. To confirm that the network plugin is OpenShift SDN, enter the following command. The value of **status.networkType** must be **OpenShiftSDN**.

```
$ oc get network.config/cluster -o jsonpath='{.status.networkType}{"\n"}'
```

b. To confirm that the cluster nodes are in the **Ready** state, enter the following command:

```
$ oc get nodes
```

c. If a node is stuck in the **NotReady** state, investigate the machine config daemon pod logs and resolve any errors.

i. To list the pods, enter the following command:

```
$ oc get pod -n openshift-machine-config-operator
```

Example output

```
NAME                                      READY   STATUS    RESTARTS   AGE
machine-config-controller-75f756f89d-sjp8b  1/1    Running   0          37m
machine-config-daemon-5cf4b                 2/2    Running   0          43h
machine-config-daemon-7wzcd                 2/2    Running   0          43h
machine-config-daemon-fc946                 2/2    Running   0          43h
machine-config-daemon-g2v28                 2/2    Running   0          43h
machine-config-daemon-gcl4f                 2/2    Running   0          43h
machine-config-daemon-l5tnv                 2/2    Running   0          43h
machine-config-operator-79d9c55d5-hth92     1/1    Running   0          37m
machine-config-server-bsc8h                 1/1    Running   0          43h
machine-config-server-hklrm                 1/1    Running   0          43h
machine-config-server-k9rtx                 1/1    Running   0          43h
```

The names for the config daemon pods are in the following format: **machine-config-daemon-<seq>**. The **<seq>** value is a random five character alphanumeric sequence.

ii. To display the pod log for each machine config daemon pod shown in the previous output, enter the following command:

```
$ oc logs <pod> -n openshift-machine-config-operator
```

where **pod** is the name of a machine config daemon pod.

  iii. Resolve any errors in the logs shown by the output from the previous command.

 d. To confirm that your pods are not in an error state, enter the following command:

```
$ oc get pods --all-namespaces -o wide --sort-by='{.spec.nodeName}'
```

If pods on a node are in an error state, reboot that node.

11. Complete the following steps only if the migration succeeds and your cluster is in a good state:

 a. To remove the migration configuration from the Cluster Network Operator configuration object, enter the following command:

```
$ oc patch Network.operator.openshift.io cluster --type='merge' \
  --patch '{ "spec": { "migration": null } }'
```

 b. To remove the OVN-Kubernetes configuration, enter the following command:

```
$ oc patch Network.operator.openshift.io cluster --type='merge' \
  --patch '{ "spec": { "defaultNetwork": { "ovnKubernetesConfig":null } } }'
```

 c. To remove the OVN-Kubernetes network provider namespace, enter the following command:

```
$ oc delete namespace openshift-ovn-kubernetes
```

28.2.3. Additional resources

- Configuration parameters for the OpenShift SDN network plugin

- Backing up etcd

- About network policy

- OpenShift SDN capabilities

  - Configuring egress IPs for a project

  - Configuring an egress firewall for a project

  - Enabling multicast for a project

- Network [operator.openshift.io/v1]

28.3. ROLLING BACK TO THE OVN-KUBERNETES NETWORK PLUGIN

As a cluster administrator, you can rollback to the OVN-Kubernetes network plugin from the OpenShift SDN network plugin if the migration to OpenShift SDN is unsuccessful.

To learn more about OVN-Kubernetes, read About the OVN-Kubernetes network plugin .

28.3.1. Migrating to the OVN-Kubernetes network plugin

As a cluster administrator, you can change the network plugin for your cluster to OVN-Kubernetes. During the migration, you must reboot every node in your cluster.

IMPORTANT

While performing the migration, your cluster is unavailable and workloads might be interrupted. Perform the migration only when an interruption in service is acceptable.

Prerequisites

- A cluster configured with the OpenShift SDN CNI network plugin in the network policy isolation mode.

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

- Access to the cluster as a user with the **cluster-admin** role.

- A recent backup of the etcd database is available.

- A reboot can be triggered manually for each node.

- The cluster is in a known good state, without any errors.

- On all cloud platforms after updating software, a security group rule must be in place to allow UDP packets on port **6081** for all nodes.

Procedure

1. To backup the configuration for the cluster network, enter the following command:

```
$ oc get Network.config.openshift.io cluster -o yaml > cluster-openshift-sdn.yaml
```

2. To prepare all the nodes for the migration, set the **migration** field on the Cluster Network Operator configuration object by entering the following command:

```
$ oc patch Network.operator.openshift.io cluster --type='merge' \
  --patch '{ "spec": { "migration": { "networkType": "OVNKubernetes" } } }'
```


NOTE

This step does not deploy OVN-Kubernetes immediately. Instead, specifying the **migration** field triggers the Machine Config Operator (MCO) to apply new machine configs to all the nodes in the cluster in preparation for the OVN-Kubernetes deployment.

3. Optional: You can disable automatic migration of several OpenShift SDN capabilities to the OVN-Kubernetes equivalents:

- Egress IPs

- Egress firewall

- Multicast

To disable automatic migration of the configuration for any of the previously noted OpenShift SDN features, specify the following keys:

```
$ oc patch Network.operator.openshift.io cluster --type='merge' \
  --patch '{
    "spec": {
      "migration": {
        "networkType": "OVNKubernetes",
        "features": {
          "egressIP": <bool>,
          "egressFirewall": <bool>,
          "multicast": <bool>
        }
      }
    }
  }'
```

where:

bool: Specifies whether to enable migration of the feature. The default is **true**.

4. Optional: You can customize the following settings for OVN-Kubernetes to meet your network infrastructure requirements:

   - Maximum transmission unit (MTU). Consider the following before customizing the MTU for this optional step:

     - If you use the default MTU, and you want to keep the default MTU during migration, this step can be ignored.

     - If you used a custom MTU, and you want to keep the custom MTU during migration, you must declare the custom MTU value in this step.

     - This step does not work if you want to change the MTU value during migration. Instead, you must first follow the instructions for "Changing the cluster MTU". You can then keep the custom MTU value by performing this procedure and declaring the custom MTU value in this step.

       > **NOTE**
       >
       > OpenShift-SDN and OVN-Kubernetes have different overlay overhead. MTU values should be selected by following the guidelines found on the "MTU value selection" page.

   - Geneve (Generic Network Virtualization Encapsulation) overlay network port

   - OVN-Kubernetes IPv4 internal subnet

   - OVN-Kubernetes IPv6 internal subnet

   To customize either of the previously noted settings, enter and customize the following command. If you do not need to change the default value, omit the key from the patch.

```
$ oc patch Network.operator.openshift.io cluster --type=merge \
  --patch '{
    "spec":{
```

```
    "defaultNetwork":{
      "ovnKubernetesConfig":{
        "mtu":<mtu>,
        "genevePort":<port>,
        "v4InternalSubnet":"<ipv4_subnet>",
        "v6InternalSubnet":"<ipv6_subnet>"
    }}}}'
```

where:

mtu

The MTU for the Geneve overlay network. This value is normally configured automatically, but if the nodes in your cluster do not all use the same MTU, then you must set this explicitly to **100** less than the smallest node MTU value.

port

The UDP port for the Geneve overlay network. If a value is not specified, the default is **6081**. The port cannot be the same as the VXLAN port that is used by OpenShift SDN. The default value for the VXLAN port is **4789**.

ipv4_subnet

An IPv4 address range for internal use by OVN-Kubernetes. You must ensure that the IP address range does not overlap with any other subnet used by your OpenShift Container Platform installation. The IP address range must be larger than the maximum number of nodes that can be added to the cluster. The default value is **100.64.0.0/16**.

ipv6_subnet

An IPv6 address range for internal use by OVN-Kubernetes. You must ensure that the IP address range does not overlap with any other subnet used by your OpenShift Container Platform installation. The IP address range must be larger than the maximum number of nodes that can be added to the cluster. The default value is **fd98::/48**.

Example patch command to update mtu field

```
$ oc patch Network.operator.openshift.io cluster --type=merge \
  --patch '{
    "spec":{
      "defaultNetwork":{
        "ovnKubernetesConfig":{
          "mtu":1200
    }}}}'
```

5. As the MCO updates machines in each machine config pool, it reboots each node one by one. You must wait until all the nodes are updated. Check the machine config pool status by entering the following command:

```
$ oc get mcp
```

A successfully updated node has the following status: **UPDATED=true**, **UPDATING=false**, **DEGRADED=false**.

> **NOTE**
>
> By default, the MCO updates one machine per pool at a time, causing the total time the migration takes to increase with the size of the cluster.

6. Confirm the status of the new machine configuration on the hosts:

   a. To list the machine configuration state and the name of the applied machine configuration, enter the following command:

   ```
   $ oc describe node | egrep "hostname|machineconfig"
   ```

   Example output

   ```
   kubernetes.io/hostname=master-0
   machineconfiguration.openshift.io/currentConfig: rendered-master-
   c53e221d9d24e1c8bb6ee89dd3d8ad7b
   machineconfiguration.openshift.io/desiredConfig: rendered-master-
   c53e221d9d24e1c8bb6ee89dd3d8ad7b
   machineconfiguration.openshift.io/reason:
   machineconfiguration.openshift.io/state: Done
   ```

   Verify that the following statements are true:

   - The value of **machineconfiguration.openshift.io/state** field is **Done**.

   - The value of the **machineconfiguration.openshift.io/currentConfig** field is equal to the value of the **machineconfiguration.openshift.io/desiredConfig** field.

   b. To confirm that the machine config is correct, enter the following command:

   ```
   $ oc get machineconfig <config_name> -o yaml | grep ExecStart
   ```

   where **<config_name>** is the name of the machine config from the **machineconfiguration.openshift.io/currentConfig** field.

   The machine config must include the following update to the systemd configuration:

   ```
   ExecStart=/usr/local/bin/configure-ovs.sh OVNKubernetes
   ```

   c. If a node is stuck in the **NotReady** state, investigate the machine config daemon pod logs and resolve any errors.

      i. To list the pods, enter the following command:

      ```
      $ oc get pod -n openshift-machine-config-operator
      ```

      Example output

      ```
      NAME                                    READY   STATUS    RESTARTS   AGE
      machine-config-controller-75f756f89d-sjp8b   1/1     Running   0          37m
      machine-config-daemon-5cf4b                  2/2     Running   0          43h
      machine-config-daemon-7wzcd                  2/2     Running   0          43h
      machine-config-daemon-fc946                  2/2     Running   0          43h
      machine-config-daemon-g2v28                  2/2     Running   0          43h
      machine-config-daemon-gcl4f                  2/2     Running   0          43h
      machine-config-daemon-l5tnv                  2/2     Running   0          43h
      machine-config-operator-79d9c55d5-hth92      1/1     Running   0          37m
      ```

```
machine-config-server-bsc8h        1/1    Running   0        43h
machine-config-server-hklrm        1/1    Running   0        43h
machine-config-server-k9rtx        1/1    Running   0        43h
```

The names for the config daemon pods are in the following format: **machine-config-daemon-<seq>**. The **<seq>** value is a random five character alphanumeric sequence.

    ii. Display the pod log for the first machine config daemon pod shown in the previous output by enter the following command:

```
$ oc logs <pod> -n openshift-machine-config-operator
```

where **pod** is the name of a machine config daemon pod.

   iii. Resolve any errors in the logs shown by the output from the previous command.

7. To start the migration, configure the OVN-Kubernetes network plugin by using one of the following commands:

- To specify the network provider without changing the cluster network IP address block, enter the following command:

```
$ oc patch Network.config.openshift.io cluster \
  --type='merge' --patch '{ "spec": { "networkType": "OVNKubernetes" } }'
```

- To specify a different cluster network IP address block, enter the following command:

```
$ oc patch Network.config.openshift.io cluster \
  --type='merge' --patch '{
    "spec": {
      "clusterNetwork": [
        {
          "cidr": "<cidr>",
          "hostPrefix": <prefix>
        }
      ],
      "networkType": "OVNKubernetes"
    }
  }'
```

where **cidr** is a CIDR block and **prefix** is the slice of the CIDR block apportioned to each node in your cluster. You cannot use any CIDR block that overlaps with the **100.64.0.0/16** CIDR block because the OVN-Kubernetes network provider uses this block internally.

> **IMPORTANT**
>
> You cannot change the service network address block during the migration.

8. Verify that the Multus daemon set rollout is complete before continuing with subsequent steps:

```
$ oc -n openshift-multus rollout status daemonset/multus
```

The name of the Multus pods is in the form of **multus-<xxxxx>** where **<xxxxx>** is a random sequence of letters. It might take several moments for the pods to restart.

Example output

> Waiting for daemon set "multus" rollout to finish: 1 out of 6 new pods have been updated...
> ...
> Waiting for daemon set "multus" rollout to finish: 5 of 6 updated pods are available...
> daemon set "multus" successfully rolled out

9. To complete changing the network plugin, reboot each node in your cluster. You can reboot the nodes in your cluster with either of the following approaches:

   - With the **oc rsh** command, you can use a bash script similar to the following:

     ```
     #!/bin/bash
     readarray -t POD_NODES <<< "$(oc get pod -n openshift-machine-config-operator -o wide| grep daemon|awk '{print $1" "$7}')"

     for i in "${POD_NODES[@]}"
     do
       read -r POD NODE <<< "$i"
       until oc rsh -n openshift-machine-config-operator "$POD" chroot /rootfs shutdown -r +1
         do
           echo "cannot reboot node $NODE, retry" && sleep 3
         done
     done
     ```

   - With the **ssh** command, you can use a bash script similar to the following. The script assumes that you have configured sudo to not prompt for a password.

     ```
     #!/bin/bash

     for ip in $(oc get nodes  -o jsonpath='{.items[*].status.addresses[?(@.type=="InternalIP")].address}')
     do
         echo "reboot node $ip"
         ssh -o StrictHostKeyChecking=no core@$ip sudo shutdown -r -t 3
     done
     ```

10. Confirm that the migration succeeded:

    a. To confirm that the network plugin is OVN-Kubernetes, enter the following command. The value of **status.networkType** must be **OVNKubernetes**.

       ```
       $ oc get network.config/cluster -o jsonpath='{.status.networkType}{"\n"}'
       ```

    b. To confirm that the cluster nodes are in the **Ready** state, enter the following command:

       ```
       $ oc get nodes
       ```

    c. To confirm that your pods are not in an error state, enter the following command:

       ```
       $ oc get pods --all-namespaces -o wide --sort-by='{.spec.nodeName}'
       ```

       If pods on a node are in an error state, reboot that node.

d. To confirm that all of the cluster Operators are not in an abnormal state, enter the following command:

```
$ oc get co
```

The status of every cluster Operator must be the following: **AVAILABLE="True"**, **PROGRESSING="False"**, **DEGRADED="False"**. If a cluster Operator is not available or degraded, check the logs for the cluster Operator for more information.

11. Complete the following steps only if the migration succeeds and your cluster is in a good state:

a. To remove the migration configuration from the CNO configuration object, enter the following command:

```
$ oc patch Network.operator.openshift.io cluster --type='merge' \
  --patch '{ "spec": { "migration": null } }'
```

b. To remove custom configuration for the OpenShift SDN network provider, enter the following command:

```
$ oc patch Network.operator.openshift.io cluster --type='merge' \
  --patch '{ "spec": { "defaultNetwork": { "openshiftSDNConfig": null } } }'
```

c. To remove the OpenShift SDN network provider namespace, enter the following command:

```
$ oc delete namespace openshift-sdn
```

28.4. CONFIGURING EGRESS IPS FOR A PROJECT

As a cluster administrator, you can configure the OpenShift SDN Container Network Interface (CNI) network plugin to assign one or more egress IP addresses to a project.

28.4.1. Egress IP address architectural design and implementation

The OpenShift Container Platform egress IP address functionality allows you to ensure that the traffic from one or more pods in one or more namespaces has a consistent source IP address for services outside the cluster network.

For example, you might have a pod that periodically queries a database that is hosted on a server outside of your cluster. To enforce access requirements for the server, a packet filtering device is configured to allow traffic only from specific IP addresses. To ensure that you can reliably allow access to the server from only that specific pod, you can configure a specific egress IP address for the pod that makes the requests to the server.

An egress IP address assigned to a namespace is different from an egress router, which is used to send traffic to specific destinations.

In some cluster configurations, application pods and ingress router pods run on the same node. If you configure an egress IP address for an application project in this scenario, the IP address is not used when you send a request to a route from the application project.

An egress IP address is implemented as an additional IP address on the primary network interface of a node and must be in the same subnet as the primary IP address of the node. The additional IP address must not be assigned to any other node in the cluster.

IMPORTANT

Egress IP addresses must not be configured in any Linux network configuration files, such as **ifcfg-eth0**.

28.4.1.1. Platform support

Support for the egress IP address functionality on various platforms is summarized in the following table:

Platform	Supported
Bare metal	Yes
VMware vSphere	Yes
Red Hat OpenStack Platform (RHOSP)	Yes
Amazon Web Services (AWS)	Yes
Google Cloud Platform (GCP)	Yes
Microsoft Azure	Yes
IBM Z and IBM® LinuxONE	Yes
IBM Z and IBM® LinuxONE for Red Hat Enterprise Linux (RHEL) KVM	Yes
IBM Power	Yes

IMPORTANT

The assignment of egress IP addresses to control plane nodes with the EgressIP feature is not supported on a cluster provisioned on Amazon Web Services (AWS). (BZ#2039656)

28.4.1.2. Public cloud platform considerations

For clusters provisioned on public cloud infrastructure, there is a constraint on the absolute number of assignable IP addresses per node. The maximum number of assignable IP addresses per node, or the *IP capacity*, can be described in the following formula:

> IP capacity = public cloud default capacity - sum(current IP assignments)

While the Egress IPs capability manages the IP address capacity per node, it is important to plan for this constraint in your deployments. For example, for a cluster installed on bare-metal infrastructure with 8 nodes you can configure 150 egress IP addresses. However, if a public cloud provider limits IP address

capacity to 10 IP addresses per node, the total number of assignable IP addresses is only 80. To achieve the same IP address capacity in this example cloud provider, you would need to allocate 7 additional nodes.

To confirm the IP capacity and subnets for any node in your public cloud environment, you can enter the **oc get node <node_name> -o yaml** command. The **cloud.network.openshift.io/egress-ipconfig** annotation includes capacity and subnet information for the node.

The annotation value is an array with a single object with fields that provide the following information for the primary network interface:

- **interface**: Specifies the interface ID on AWS and Azure and the interface name on GCP.

- **ifaddr**: Specifies the subnet mask for one or both IP address families.

- **capacity**: Specifies the IP address capacity for the node. On AWS, the IP address capacity is provided per IP address family. On Azure and GCP, the IP address capacity includes both IPv4 and IPv6 addresses.

Automatic attachment and detachment of egress IP addresses for traffic between nodes are available. This allows for traffic from many pods in namespaces to have a consistent source IP address to locations outside of the cluster. This also supports OpenShift SDN and OVN-Kubernetes, which is the default networking plugin in Red Hat OpenShift Networking in OpenShift Container Platform 4.13.

> **NOTE**
>
> The RHOSP egress IP address feature creates a Neutron reservation port called **egressip-<IP address>**. Using the same RHOSP user as the one used for the OpenShift Container Platform cluster installation, you can assign a floating IP address to this reservation port to have a predictable SNAT address for egress traffic. When an egress IP address on an RHOSP network is moved from one node to another, because of a node failover, for example, the Neutron reservation port is removed and recreated. This means that the floating IP association is lost and you need to manually reassign the floating IP address to the new reservation port.

> **NOTE**
>
> When an RHOSP cluster administrator assigns a floating IP to the reservation port, OpenShift Container Platform cannot delete the reservation port. The **CloudPrivateIPConfig** object cannot perform delete and move operations until an RHOSP cluster administrator unassigns the floating IP from the reservation port.

The following examples illustrate the annotation from nodes on several public cloud providers. The annotations are indented for readability.

Example cloud.network.openshift.io/egress-ipconfig annotation on AWS

```
cloud.network.openshift.io/egress-ipconfig: [
  {
    "interface":"eni-078d267045138e436",
    "ifaddr":{"ipv4":"10.0.128.0/18"},
    "capacity":{"ipv4":14,"ipv6":15}
  }
]
```

Example cloud.network.openshift.io/egress-ipconfig annotation on GCP

```
cloud.network.openshift.io/egress-ipconfig: [
  {
    "interface":"nic0",
    "ifaddr":{"ipv4":"10.0.128.0/18"},
    "capacity":{"ip":14}
  }
]
```

The following sections describe the IP address capacity for supported public cloud environments for use in your capacity calculation.

28.4.1.2.1. Amazon Web Services (AWS) IP address capacity limits

On AWS, constraints on IP address assignments depend on the instance type configured. For more information, see IP addresses per network interface per instance type

28.4.1.2.2. Google Cloud Platform (GCP) IP address capacity limits

On GCP, the networking model implements additional node IP addresses through IP address aliasing, rather than IP address assignments. However, IP address capacity maps directly to IP aliasing capacity.

The following capacity limits exist for IP aliasing assignment:

- Per node, the maximum number of IP aliases, both IPv4 and IPv6, is 100.

- Per VPC, the maximum number of IP aliases is unspecified, but OpenShift Container Platform scalability testing reveals the maximum to be approximately 15,000.

For more information, see Per instance quotas and Alias IP ranges overview .

28.4.1.2.3. Microsoft Azure IP address capacity limits

On Azure, the following capacity limits exist for IP address assignment:

- Per NIC, the maximum number of assignable IP addresses, for both IPv4 and IPv6, is 256.

- Per virtual network, the maximum number of assigned IP addresses cannot exceed 65,536.

For more information, see Networking limits .

28.4.1.3. Limitations

The following limitations apply when using egress IP addresses with the OpenShift SDN network plugin:

- You cannot use manually assigned and automatically assigned egress IP addresses on the same nodes.

- If you manually assign egress IP addresses from an IP address range, you must not make that range available for automatic IP assignment.

- You cannot share egress IP addresses across multiple namespaces using the OpenShift SDN egress IP address implementation.

If you need to share IP addresses across namespaces, the OVN-Kubernetes network plugin egress IP address implementation allows you to span IP addresses across multiple namespaces.

> **NOTE**
>
> If you use OpenShift SDN in multitenant mode, you cannot use egress IP addresses with any namespace that is joined to another namespace by the projects that are associated with them. For example, if **project1** and **project2** are joined by running the **oc adm pod-network join-projects --to=project1 project2** command, neither project can use an egress IP address. For more information, see BZ#1645577.

28.4.1.4. IP address assignment approaches

You can assign egress IP addresses to namespaces by setting the **egressIPs** parameter of the **NetNamespace** object. After an egress IP address is associated with a project, OpenShift SDN allows you to assign egress IP addresses to hosts in two ways:

- In the *automatically assigned* approach, an egress IP address range is assigned to a node.

- In the *manually assigned* approach, a list of one or more egress IP address is assigned to a node.

Namespaces that request an egress IP address are matched with nodes that can host those egress IP addresses, and then the egress IP addresses are assigned to those nodes. If the **egressIPs** parameter is set on a **NetNamespace** object, but no node hosts that egress IP address, then egress traffic from the namespace will be dropped.

High availability of nodes is automatic. If a node that hosts an egress IP address is unreachable and there are nodes that are able to host that egress IP address, then the egress IP address will move to a new node. When the unreachable node comes back online, the egress IP address automatically moves to balance egress IP addresses across nodes.

28.4.1.4.1. Considerations when using automatically assigned egress IP addresses

When using the automatic assignment approach for egress IP addresses the following considerations apply:

- You set the **egressCIDRs** parameter of each node's **HostSubnet** resource to indicate the range of egress IP addresses that can be hosted by a node. OpenShift Container Platform sets the **egressIPs** parameter of the **HostSubnet** resource based on the IP address range you specify.

If the node hosting the namespace's egress IP address is unreachable, OpenShift Container Platform will reassign the egress IP address to another node with a compatible egress IP address range. The automatic assignment approach works best for clusters installed in environments with flexibility in associating additional IP addresses with nodes.

28.4.1.4.2. Considerations when using manually assigned egress IP addresses

This approach allows you to control which nodes can host an egress IP address.

> **NOTE**
>
> If your cluster is installed on public cloud infrastructure, you must ensure that each node that you assign egress IP addresses to has sufficient spare capacity to host the IP addresses. For more information, see "Platform considerations" in a previous section.

When using the manual assignment approach for egress IP addresses the following considerations apply:

- You set the **egressIPs** parameter of each node's **HostSubnet** resource to indicate the IP addresses that can be hosted by a node.

- Multiple egress IP addresses per namespace are supported.

If a namespace has multiple egress IP addresses and those addresses are hosted on multiple nodes, the following additional considerations apply:

- If a pod is on a node that is hosting an egress IP address, that pod always uses the egress IP address on the node.

- If a pod is not on a node that is hosting an egress IP address, that pod uses an egress IP address at random.

28.4.2. Configuring automatically assigned egress IP addresses for a namespace

In OpenShift Container Platform you can enable automatic assignment of an egress IP address for a specific namespace across one or more nodes.

Prerequisites

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

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

Procedure

1. Update the **NetNamespace** object with the egress IP address using the following JSON:

   ```
   $ oc patch netnamespace <project_name> --type=merge -p \
   '{
     "egressIPs": [
       "<ip_address>"
     ]
   }'
   ```

   where:

   <project_name>

   Specifies the name of the project.

   <ip_address>

   Specifies one or more egress IP addresses for the **egressIPs** array.

   For example, to assign **project1** to an IP address of 192.168.1.100 and **project2** to an IP address of 192.168.1.101:

   ```
   $ oc patch netnamespace project1 --type=merge -p \
     '{"egressIPs": ["192.168.1.100"]}'
   $ oc patch netnamespace project2 --type=merge -p \
     '{"egressIPs": ["192.168.1.101"]}'
   ```

> **NOTE**
>
> Because OpenShift SDN manages the **NetNamespace** object, you can make changes only by modifying the existing **NetNamespace** object. Do not create a new **NetNamespace** object.

2. Indicate which nodes can host egress IP addresses by setting the **egressCIDRs** parameter for each host using the following JSON:

```
$ oc patch hostsubnet <node_name> --type=merge -p \
 '{
   "egressCIDRs": [
     "<ip_address_range>", "<ip_address_range>"
   ]
 }'
```

where:

<node_name>

    Specifies a node name.

<ip_address_range>

    Specifies an IP address range in CIDR format. You can specify more than one address range for the **egressCIDRs** array.

For example, to set **node1** and **node2** to host egress IP addresses in the range 192.168.1.0 to 192.168.1.255:

```
$ oc patch hostsubnet node1 --type=merge -p \
  '{"egressCIDRs": ["192.168.1.0/24"]}'
$ oc patch hostsubnet node2 --type=merge -p \
  '{"egressCIDRs": ["192.168.1.0/24"]}'
```

OpenShift Container Platform automatically assigns specific egress IP addresses to available nodes in a balanced way. In this case, it assigns the egress IP address 192.168.1.100 to **node1** and the egress IP address 192.168.1.101 to **node2** or vice versa.

28.4.3. Configuring manually assigned egress IP addresses for a namespace

In OpenShift Container Platform you can associate one or more egress IP addresses with a namespace.

Prerequisites

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

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

Procedure

1. Update the **NetNamespace** object by specifying the following JSON object with the desired IP addresses:

```
$ oc patch netnamespace <project_name> --type=merge -p \
 '{
```

```
  "egressIPs": [
    "<ip_address>"
  ]
}'
```

where:

<project_name>

Specifies the name of the project.

<ip_address>

Specifies one or more egress IP addresses for the **egressIPs** array.

For example, to assign the **project1** project to the IP addresses **192.168.1.100** and **192.168.1.101**:

```
$ oc patch netnamespace project1 --type=merge \
  -p '{"egressIPs": ["192.168.1.100","192.168.1.101"]}'
```

To provide high availability, set the **egressIPs** value to two or more IP addresses on different nodes. If multiple egress IP addresses are set, then pods use all egress IP addresses roughly equally.

> **NOTE**
>
> Because OpenShift SDN manages the **NetNamespace** object, you can make changes only by modifying the existing **NetNamespace** object. Do not create a new **NetNamespace** object.

2. Manually assign the egress IP address to the node hosts.
   If your cluster is installed on public cloud infrastructure, you must confirm that the node has available IP address capacity.

   Set the **egressIPs** parameter on the **HostSubnet** object on the node host. Using the following JSON, include as many IP addresses as you want to assign to that node host:

   ```
   $ oc patch hostsubnet <node_name> --type=merge -p \
     '{
       "egressIPs": [
         "<ip_address>",
         "<ip_address>"
       ]
     }'
   ```

   where:

   <node_name>

   Specifies a node name.

   <ip_address>

   Specifies an IP address. You can specify more than one IP address for the **egressIPs** array.

   For example, to specify that **node1** should have the egress IPs **192.168.1.100**, **192.168.1.101**, and **192.168.1.102**:

```
$ oc patch hostsubnet node1 --type=merge -p \
  '{"egressIPs": ["192.168.1.100", "192.168.1.101", "192.168.1.102"]}'
```

In the previous example, all egress traffic for **project1** will be routed to the node hosting the specified egress IP, and then connected through Network Address Translation (NAT) to that IP address.

28.4.4. Additional resources

- If you are configuring manual egress IP address assignment, see Platform considerations for information about IP capacity planning.

28.5. CONFIGURING AN EGRESS FIREWALL FOR A PROJECT

As a cluster administrator, you can create an egress firewall for a project that restricts egress traffic leaving your OpenShift Container Platform cluster.

28.5.1. How an egress firewall works in a project

As a cluster administrator, you can use an *egress firewall* to limit the external hosts that some or all pods can access from within the cluster. An egress firewall supports the following scenarios:

- A pod can only connect to internal hosts and cannot initiate connections to the public internet.

- A pod can only connect to the public internet and cannot initiate connections to internal hosts that are outside the OpenShift Container Platform cluster.

- A pod cannot reach specified internal subnets or hosts outside the OpenShift Container Platform cluster.

- A pod can connect to only specific external hosts.

For example, you can allow one project access to a specified IP range but deny the same access to a different project. Or you can restrict application developers from updating from Python pip mirrors, and force updates to come only from approved sources.

> **NOTE**
>
> Egress firewall does not apply to the host network namespace. Pods with host networking enabled are unaffected by egress firewall rules.

You configure an egress firewall policy by creating an EgressNetworkPolicy custom resource (CR) object. The egress firewall matches network traffic that meets any of the following criteria:

- An IP address range in CIDR format

- A DNS name that resolves to an IP address

IMPORTANT

If your egress firewall includes a deny rule for **0.0.0.0/0**, access to your OpenShift Container Platform API servers is blocked. You must either add allow rules for each IP address or use the **nodeSelector** type allow rule in your egress policy rules to connect to API servers.

The following example illustrates the order of the egress firewall rules necessary to ensure API server access:

```
apiVersion: network.openshift.io/v1
kind: EgressNetworkPolicy
metadata:
  name: default
  namespace: <namespace> 1
spec:
  egress:
  - to:
      cidrSelector: <api_server_address_range> 2
    type: Allow
# ...
  - to:
      cidrSelector: 0.0.0.0/0 3
    type: Deny
```

1   The namespace for the egress firewall.

2   The IP address range that includes your OpenShift Container Platform API servers.

3   A global deny rule prevents access to the OpenShift Container Platform API servers.

To find the IP address for your API servers, run **oc get ep kubernetes -n default**.

For more information, see BZ#1988324.

IMPORTANT

You must have OpenShift SDN configured to use either the network policy or multitenant mode to configure an egress firewall.

If you use network policy mode, an egress firewall is compatible with only one policy per namespace and will not work with projects that share a network, such as global projects.

WARNING

Egress firewall rules do not apply to traffic that goes through routers. Any user with permission to create a Route CR object can bypass egress firewall policy rules by creating a route that points to a forbidden destination.

28.5.1.1. Limitations of an egress firewall

An egress firewall has the following limitations:

- No project can have more than one EgressNetworkPolicy object.

> **IMPORTANT**
>
> The creation of more than one EgressNetworkPolicy object is allowed, however it should not be done. When you create more than one EgressNetworkPolicy object, the following message is returned: **dropping all rules**. In actuality, all external traffic is dropped, which can cause security risks for your organization.

- A maximum of one EgressNetworkPolicy object with a maximum of 1,000 rules can be defined per project.

- The **default** project cannot use an egress firewall.

- When using the OpenShift SDN network plugin in multitenant mode, the following limitations apply:

  - Global projects cannot use an egress firewall. You can make a project global by using the **oc adm pod-network make-projects-global** command.

  - Projects merged by using the **oc adm pod-network join-projects** command cannot use an egress firewall in any of the joined projects.

Violating any of these restrictions results in a broken egress firewall for the project. Consequently, all external network traffic is dropped, which can cause security risks for your organization.

An Egress Firewall resource can be created in the **kube-node-lease**, **kube-public**, **kube-system**, **openshift** and **openshift-** projects.

28.5.1.2. Matching order for egress firewall policy rules

The egress firewall policy rules are evaluated in the order that they are defined, from first to last. The first rule that matches an egress connection from a pod applies. Any subsequent rules are ignored for that connection.

28.5.1.3. How Domain Name Server (DNS) resolution works

If you use DNS names in any of your egress firewall policy rules, proper resolution of the domain names is subject to the following restrictions:

- Domain name updates are polled based on a time-to-live (TTL) duration. By default, the duration is 30 seconds. When the egress firewall controller queries the local name servers for a domain name, if the response includes a TTL that is less than 30 seconds, the controller sets the duration to the returned value. If the TTL in the response is greater than 30 minutes, the controller sets the duration to 30 minutes. If the TTL is between 30 seconds and 30 minutes, the controller ignores the value and sets the duration to 30 seconds.

- The pod must resolve the domain from the same local name servers when necessary. Otherwise the IP addresses for the domain known by the egress firewall controller and the pod can be different. If the IP addresses for a hostname differ, the egress firewall might not be enforced consistently.

- Because the egress firewall controller and pods asynchronously poll the same local name server, the pod might obtain the updated IP address before the egress controller does, which causes a race condition. Due to this current limitation, domain name usage in EgressNetworkPolicy objects is only recommended for domains with infrequent IP address changes.

> **NOTE**
>
> Using DNS names in your egress firewall policy does not affect local DNS resolution through CoreDNS.
>
> However, if your egress firewall policy uses domain names, and an external DNS server handles DNS resolution for an affected pod, you must include egress firewall rules that permit access to the IP addresses of your DNS server.

28.5.2. EgressNetworkPolicy custom resource (CR) object

You can define one or more rules for an egress firewall. A rule is either an **Allow** rule or a **Deny** rule, with a specification for the traffic that the rule applies to.

The following YAML describes an EgressNetworkPolicy CR object:

EgressNetworkPolicy object

```
apiVersion: network.openshift.io/v1
kind: EgressNetworkPolicy
metadata:
  name: <name> 1
spec:
  egress: 2
    ...
```

1 A name for your egress firewall policy.

2 A collection of one or more egress network policy rules as described in the following section.

28.5.2.1. EgressNetworkPolicy rules

The following YAML describes an egress firewall rule object. The user can select either an IP address range in CIDR format, a domain name, or use the **nodeSelector** to allow or deny egress traffic. The **egress** stanza expects an array of one or more objects.

Egress policy rule stanza

```
egress:
- type: <type> 1
  to: 2
    cidrSelector: <cidr> 3
    dnsName: <dns_name> 4
```

1 The type of rule. The value must be either **Allow** or **Deny**.

2 A stanza describing an egress traffic match rule. A value for either the **cidrSelector** field or the **dnsName** field for the rule. You cannot use both fields in the same rule.

3 An IP address range in CIDR format.

4 A domain name.

28.5.2.2. Example EgressNetworkPolicy CR objects

The following example defines several egress firewall policy rules:

```
apiVersion: network.openshift.io/v1
kind: EgressNetworkPolicy
metadata:
  name: default
spec:
  egress: 1
  - type: Allow
    to:
      cidrSelector: 1.2.3.0/24
  - type: Allow
    to:
      dnsName: www.example.com
  - type: Deny
    to:
      cidrSelector: 0.0.0.0/0
```

1 A collection of egress firewall policy rule objects.

28.5.3. Creating an egress firewall policy object

As a cluster administrator, you can create an egress firewall policy object for a project.

> **IMPORTANT**
>
> If the project already has an EgressNetworkPolicy object defined, you must edit the existing policy to make changes to the egress firewall rules.

Prerequisites

- A cluster that uses the OpenShift SDN network plugin.

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

- You must log in to the cluster as a cluster administrator.

Procedure

1. Create a policy rule:

    a. Create a **<policy_name>.yaml** file where **<policy_name>** describes the egress policy rules.

    b. In the file you created, define an egress policy object.

2. Enter the following command to create the policy object. Replace **<policy_name>** with the name of the policy and **<project>** with the project that the rule applies to.

    ```
    $ oc create -f <policy_name>.yaml -n <project>
    ```

    In the following example, a new EgressNetworkPolicy object is created in a project named **project1**:

```
$ oc create -f default.yaml -n project1
```

Example output

```
egressnetworkpolicy.network.openshift.io/v1 created
```

3. Optional: Save the **<policy_name>.yaml** file so that you can make changes later.

28.6. EDITING AN EGRESS FIREWALL FOR A PROJECT

As a cluster administrator, you can modify network traffic rules for an existing egress firewall.

28.6.1. Viewing an EgressNetworkPolicy object

You can view an EgressNetworkPolicy object in your cluster.

Prerequisites

- A cluster using the OpenShift SDN network plugin.

- Install the OpenShift Command-line Interface (CLI), commonly known as **oc**.

- You must log in to the cluster.

Procedure

1. Optional: To view the names of the EgressNetworkPolicy objects defined in your cluster, enter the following command:

```
$ oc get egressnetworkpolicy --all-namespaces
```

2. To inspect a policy, enter the following command. Replace **<policy_name>** with the name of the policy to inspect.

```
$ oc describe egressnetworkpolicy <policy_name>
```

Example output

```
Name:  default
Namespace: project1
Created: 20 minutes ago
Labels:  <none>
Annotations: <none>
Rule:  Allow to 1.2.3.0/24
Rule:  Allow to www.example.com
Rule:  Deny to 0.0.0.0/0
```

28.7. EDITING AN EGRESS FIREWALL FOR A PROJECT

As a cluster administrator, you can modify network traffic rules for an existing egress firewall.

28.7.1. Editing an EgressNetworkPolicy object

As a cluster administrator, you can update the egress firewall for a project.

Prerequisites

- A cluster using the OpenShift SDN network plugin.

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

- You must log in to the cluster as a cluster administrator.

Procedure

1. Find the name of the EgressNetworkPolicy object for the project. Replace **<project>** with the name of the project.

   ```
   $ oc get -n <project> egressnetworkpolicy
   ```

2. Optional: If you did not save a copy of the EgressNetworkPolicy object when you created the egress network firewall, enter the following command to create a copy.

   ```
   $ oc get -n <project> egressnetworkpolicy <name> -o yaml > <filename>.yaml
   ```

   Replace **<project>** with the name of the project. Replace **<name>** with the name of the object. Replace **<filename>** with the name of the file to save the YAML to.

3. After making changes to the policy rules, enter the following command to replace the EgressNetworkPolicy object. Replace **<filename>** with the name of the file containing the updated EgressNetworkPolicy object.

   ```
   $ oc replace -f <filename>.yaml
   ```

28.8. REMOVING AN EGRESS FIREWALL FROM A PROJECT

As a cluster administrator, you can remove an egress firewall from a project to remove all restrictions on network traffic from the project that leaves the OpenShift Container Platform cluster.

28.8.1. Removing an EgressNetworkPolicy object

As a cluster administrator, you can remove an egress firewall from a project.

Prerequisites

- A cluster using the OpenShift SDN network plugin.

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

- You must log in to the cluster as a cluster administrator.

Procedure

1. Find the name of the EgressNetworkPolicy object for the project. Replace **<project>** with the name of the project.

```
$ oc get -n <project> egressnetworkpolicy
```

2. Enter the following command to delete the EgressNetworkPolicy object. Replace **<project>** with the name of the project and **<name>** with the name of the object.

```
$ oc delete -n <project> egressnetworkpolicy <name>
```

28.9. CONSIDERATIONS FOR THE USE OF AN EGRESS ROUTER POD

28.9.1. About an egress router pod

The OpenShift Container Platform egress router pod redirects traffic to a specified remote server from a private source IP address that is not used for any other purpose. An egress router pod can send network traffic to servers that are set up to allow access only from specific IP addresses.

NOTE

The egress router pod is not intended for every outgoing connection. Creating large numbers of egress router pods can exceed the limits of your network hardware. For example, creating an egress router pod for every project or application could exceed the number of local MAC addresses that the network interface can handle before reverting to filtering MAC addresses in software.

IMPORTANT

The egress router image is not compatible with Amazon AWS, Azure Cloud, or any other cloud platform that does not support layer 2 manipulations due to their incompatibility with macvlan traffic.

28.9.1.1. Egress router modes

In *redirect mode*, an egress router pod configures **iptables** rules to redirect traffic from its own IP address to one or more destination IP addresses. Client pods that need to use the reserved source IP address must be configured to access the service for the egress router rather than connecting directly to the destination IP. You can access the destination service and port from the application pod by using the **curl** command. For example:

```
$ curl <router_service_IP> <port>
```

In *HTTP proxy mode*, an egress router pod runs as an HTTP proxy on port **8080**. This mode only works for clients that are connecting to HTTP-based or HTTPS-based services, but usually requires fewer changes to the client pods to get them to work. Many programs can be told to use an HTTP proxy by setting an environment variable.

In *DNS proxy mode*, an egress router pod runs as a DNS proxy for TCP-based services from its own IP address to one or more destination IP addresses. To make use of the reserved, source IP address, client pods must be modified to connect to the egress router pod rather than connecting directly to the destination IP address. This modification ensures that external destinations treat traffic as though it were coming from a known source.

Redirect mode works for all services except for HTTP and HTTPS. For HTTP and HTTPS services, use HTTP proxy mode. For TCP-based services with IP addresses or domain names, use DNS proxy mode.

28.9.1.2. Egress router pod implementation

The egress router pod setup is performed by an initialization container. That container runs in a privileged context so that it can configure the macvlan interface and set up **iptables** rules. After the initialization container finishes setting up the **iptables** rules, it exits. Next the egress router pod executes the container to handle the egress router traffic. The image used varies depending on the egress router mode.

The environment variables determine which addresses the egress-router image uses. The image configures the macvlan interface to use **EGRESS_SOURCE** as its IP address, with **EGRESS_GATEWAY** as the IP address for the gateway.

Network Address Translation (NAT) rules are set up so that connections to the cluster IP address of the pod on any TCP or UDP port are redirected to the same port on IP address specified by the **EGRESS_DESTINATION** variable.

If only some of the nodes in your cluster are capable of claiming the specified source IP address and using the specified gateway, you can specify a **nodeName** or **nodeSelector** to identify which nodes are acceptable.

28.9.1.3. Deployment considerations

An egress router pod adds an additional IP address and MAC address to the primary network interface of the node. As a result, you might need to configure your hypervisor or cloud provider to allow the additional address.

Red Hat OpenStack Platform (RHOSP)

If you deploy OpenShift Container Platform on RHOSP, you must allow traffic from the IP and MAC addresses of the egress router pod on your OpenStack environment. If you do not allow the traffic, then communication will fail:

```
$ openstack port set --allowed-address \
  ip_address=<ip_address>,mac_address=<mac_address> <neutron_port_uuid>
```

Red Hat Virtualization (RHV)

If you are using RHV, you must select **No Network Filter** for the Virtual network interface controller (vNIC).

VMware vSphere

If you are using VMware vSphere, see the VMware documentation for securing vSphere standard switches. View and change VMware vSphere default settings by selecting the host virtual switch from the vSphere Web Client.

Specifically, ensure that the following are enabled:

- MAC Address Changes

- Forged Transits

- Promiscuous Mode Operation

28.9.1.4. Failover configuration

To avoid downtime, you can deploy an egress router pod with a **Deployment** resource, as in the following example. To create a new **Service** object for the example deployment, use the **oc expose deployment/egress-demo-controller** command.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: egress-demo-controller
spec:
  replicas: 1 1
  selector:
    matchLabels:
      name: egress-router
  template:
    metadata:
      name: egress-router
      labels:
        name: egress-router
      annotations:
        pod.network.openshift.io/assign-macvlan: "true"
    spec: 2
      initContainers:
        ...
      containers:
        ...
```

1 Ensure that replicas is set to **1**, because only one pod can use a given egress source IP address at any time. This means that only a single copy of the router runs on a node.

2 Specify the **Pod** object template for the egress router pod.

28.9.2. Additional resources

- Deploying an egress router in redirection mode

- Deploying an egress router in HTTP proxy mode

- Deploying an egress router in DNS proxy mode

28.10. DEPLOYING AN EGRESS ROUTER POD IN REDIRECT MODE

As a cluster administrator, you can deploy an egress router pod that is configured to redirect traffic to specified destination IP addresses.

28.10.1. Egress router pod specification for redirect mode

Define the configuration for an egress router pod in the **Pod** object. The following YAML describes the fields for the configuration of an egress router pod in redirect mode:

```
apiVersion: v1
kind: Pod
metadata:
  name: egress-1
```

```
    labels:
      name: egress-1
    annotations:
      pod.network.openshift.io/assign-macvlan: "true" 1
spec:
  initContainers:
  - name: egress-router
    image: registry.redhat.io/openshift4/ose-egress-router
    securityContext:
      privileged: true
    env:
    - name: EGRESS_SOURCE 2
      value: <egress_router>
    - name: EGRESS_GATEWAY 3
      value: <egress_gateway>
    - name: EGRESS_DESTINATION 4
      value: <egress_destination>
    - name: EGRESS_ROUTER_MODE
      value: init
  containers:
  - name: egress-router-wait
    image: registry.redhat.io/openshift4/ose-pod
```

[1] The annotation tells OpenShift Container Platform to create a macvlan network interface on the primary network interface controller (NIC) and move that macvlan interface into the pod's network namespace. You must include the quotation marks around the **"true"** value. To have OpenShift Container Platform create the macvlan interface on a different NIC interface, set the annotation value to the name of that interface. For example, **eth1**.

[2] IP address from the physical network that the node is on that is reserved for use by the egress router pod. Optional: You can include the subnet length, the **/24** suffix, so that a proper route to the local subnet is set. If you do not specify a subnet length, then the egress router can access only the host specified with the **EGRESS_GATEWAY** variable and no other hosts on the subnet.

[3] Same value as the default gateway used by the node.

[4] External server to direct traffic to. Using this example, connections to the pod are redirected to **203.0.113.25**, with a source IP address of **192.168.12.99**.

Example egress router pod specification

```
apiVersion: v1
kind: Pod
metadata:
  name: egress-multi
  labels:
    name: egress-multi
  annotations:
    pod.network.openshift.io/assign-macvlan: "true"
spec:
  initContainers:
  - name: egress-router
    image: registry.redhat.io/openshift4/ose-egress-router
    securityContext:
```

```
      privileged: true
    env:
    - name: EGRESS_SOURCE
      value: 192.168.12.99/24
    - name: EGRESS_GATEWAY
      value: 192.168.12.1
    - name: EGRESS_DESTINATION
      value: |
        80   tcp 203.0.113.25
        8080 tcp 203.0.113.26 80
        8443 tcp 203.0.113.26 443
        203.0.113.27
    - name: EGRESS_ROUTER_MODE
      value: init
  containers:
  - name: egress-router-wait
    image: registry.redhat.io/openshift4/ose-pod
```

28.10.2. Egress destination configuration format

When an egress router pod is deployed in redirect mode, you can specify redirection rules by using one or more of the following formats:

- **<port> <protocol> <ip_address>** – Incoming connections to the given **<port>** should be redirected to the same port on the given **<ip_address>**. **<protocol>** is either **tcp** or **udp**.

- **<port> <protocol> <ip_address> <remote_port>** – As above, except that the connection is redirected to a different **<remote_port>** on **<ip_address>**.

- **<ip_address>** – If the last line is a single IP address, then any connections on any other port will be redirected to the corresponding port on that IP address. If there is no fallback IP address then connections on other ports are rejected.

In the example that follows several rules are defined:

- The first line redirects traffic from local port **80** to port **80** on **203.0.113.25**.

- The second and third lines redirect local ports **8080** and **8443** to remote ports **80** and **443** on **203.0.113.26**.

- The last line matches traffic for any ports not specified in the previous rules.

Example configuration

```
80   tcp 203.0.113.25
8080 tcp 203.0.113.26 80
8443 tcp 203.0.113.26 443
203.0.113.27
```

28.10.3. Deploying an egress router pod in redirect mode

In *redirect mode*, an egress router pod sets up iptables rules to redirect traffic from its own IP address to one or more destination IP addresses. Client pods that need to use the reserved source IP address must be configured to access the service for the egress router rather than connecting directly to the

destination IP. You can access the destination service and port from the application pod by using the **curl** command. For example:

```
$ curl <router_service_IP> <port>
```

Prerequisites

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

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

Procedure

1. Create an egress router pod.

2. To ensure that other pods can find the IP address of the egress router pod, create a service to point to the egress router pod, as in the following example:

```
apiVersion: v1
kind: Service
metadata:
  name: egress-1
spec:
  ports:
  - name: http
    port: 80
  - name: https
    port: 443
  type: ClusterIP
  selector:
    name: egress-1
```

Your pods can now connect to this service. Their connections are redirected to the corresponding ports on the external server, using the reserved egress IP address.

28.10.4. Additional resources

- Configuring an egress router destination mappings with a ConfigMap

28.11. DEPLOYING AN EGRESS ROUTER POD IN HTTP PROXY MODE

As a cluster administrator, you can deploy an egress router pod configured to proxy traffic to specified HTTP and HTTPS-based services.

28.11.1. Egress router pod specification for HTTP mode

Define the configuration for an egress router pod in the **Pod** object. The following YAML describes the fields for the configuration of an egress router pod in HTTP mode:

```
apiVersion: v1
kind: Pod
metadata:
  name: egress-1
```

```
    labels:
      name: egress-1
    annotations:
      pod.network.openshift.io/assign-macvlan: "true"  1
spec:
  initContainers:
  - name: egress-router
    image: registry.redhat.io/openshift4/ose-egress-router
    securityContext:
      privileged: true
    env:
    - name: EGRESS_SOURCE  2
      value: <egress-router>
    - name: EGRESS_GATEWAY  3
      value: <egress-gateway>
    - name: EGRESS_ROUTER_MODE
      value: http-proxy
  containers:
  - name: egress-router-pod
    image: registry.redhat.io/openshift4/ose-egress-http-proxy
    env:
    - name: EGRESS_HTTP_PROXY_DESTINATION  4
      value: |-
        ...
    ...
```

1   The annotation tells OpenShift Container Platform to create a macvlan network interface on the primary network interface controller (NIC) and move that macvlan interface into the pod's network namespace. You must include the quotation marks around the **"true"** value. To have OpenShift Container Platform create the macvlan interface on a different NIC interface, set the annotation value to the name of that interface. For example, **eth1**.

2   IP address from the physical network that the node is on that is reserved for use by the egress router pod. Optional: You can include the subnet length, the **/24** suffix, so that a proper route to the local subnet is set. If you do not specify a subnet length, then the egress router can access only the host specified with the **EGRESS_GATEWAY** variable and no other hosts on the subnet.

3   Same value as the default gateway used by the node.

4   A string or YAML multi-line string specifying how to configure the proxy. Note that this is specified as an environment variable in the HTTP proxy container, not with the other environment variables in the init container.

28.11.2. Egress destination configuration format

When an egress router pod is deployed in HTTP proxy mode, you can specify redirection rules by using one or more of the following formats. Each line in the configuration specifies one group of connections to allow or deny:

- An IP address allows connections to that IP address, such as **192.168.1.1**.

- A CIDR range allows connections to that CIDR range, such as **192.168.1.0/24**.

- A hostname allows proxying to that host, such as **www.example.com**.

- A domain name preceded by ***.** allows proxying to that domain and all of its subdomains, such as ***.example.com**.

- A **!** followed by any of the previous match expressions denies the connection instead.

- If the last line is *****, then anything that is not explicitly denied is allowed. Otherwise, anything that is not allowed is denied.

You can also use ***** to allow connections to all remote destinations.

Example configuration

```
!*.example.com
!192.168.1.0/24
192.168.2.1
*
```

28.11.3. Deploying an egress router pod in HTTP proxy mode

In *HTTP proxy mode*, an egress router pod runs as an HTTP proxy on port **8080**. This mode only works for clients that are connecting to HTTP-based or HTTPS-based services, but usually requires fewer changes to the client pods to get them to work. Many programs can be told to use an HTTP proxy by setting an environment variable.

Prerequisites

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

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

Procedure

1. Create an egress router pod.

2. To ensure that other pods can find the IP address of the egress router pod, create a service to point to the egress router pod, as in the following example:

   ```
   apiVersion: v1
   kind: Service
   metadata:
     name: egress-1
   spec:
     ports:
     - name: http-proxy
       port: 8080 ❶
     type: ClusterIP
     selector:
       name: egress-1
   ```

   ❶ Ensure the **http** port is set to **8080**.

3. To configure the client pod (not the egress proxy pod) to use the HTTP proxy, set the **http_proxy** or **https_proxy** variables:

```
apiVersion: v1
kind: Pod
metadata:
  name: app-1
  labels:
    name: app-1
spec:
  containers:
    env:
    - name: http_proxy
      value: http://egress-1:8080/ 1
    - name: https_proxy
      value: http://egress-1:8080/
    ...
```

1   The service created in the previous step.

> **NOTE**
>
> Using the **http_proxy** and **https_proxy** environment variables is not necessary for all setups. If the above does not create a working setup, then consult the documentation for the tool or software you are running in the pod.

28.11.4. Additional resources

- Configuring an egress router destination mappings with a ConfigMap

28.12. DEPLOYING AN EGRESS ROUTER POD IN DNS PROXY MODE

As a cluster administrator, you can deploy an egress router pod configured to proxy traffic to specified DNS names and IP addresses.

28.12.1. Egress router pod specification for DNS mode

Define the configuration for an egress router pod in the **Pod** object. The following YAML describes the fields for the configuration of an egress router pod in DNS mode:

```
apiVersion: v1
kind: Pod
metadata:
  name: egress-1
  labels:
    name: egress-1
  annotations:
    pod.network.openshift.io/assign-macvlan: "true" 1
spec:
  initContainers:
  - name: egress-router
    image: registry.redhat.io/openshift4/ose-egress-router
    securityContext:
      privileged: true
    env:
```

```
    - name: EGRESS_SOURCE 2
      value: <egress-router>
    - name: EGRESS_GATEWAY 3
      value: <egress-gateway>
    - name: EGRESS_ROUTER_MODE
      value: dns-proxy
  containers:
  - name: egress-router-pod
    image: registry.redhat.io/openshift4/ose-egress-dns-proxy
    securityContext:
      privileged: true
    env:
    - name: EGRESS_DNS_PROXY_DESTINATION 4
      value: |-
        ...
    - name: EGRESS_DNS_PROXY_DEBUG 5
      value: "1"
    ...
```

1	The annotation tells OpenShift Container Platform to create a macvlan network interface on the primary network interface controller (NIC) and move that macvlan interface into the pod's network namespace. You must include the quotation marks around the **"true"** value. To have OpenShift Container Platform create the macvlan interface on a different NIC interface, set the annotation value to the name of that interface. For example, **eth1**.
2	IP address from the physical network that the node is on that is reserved for use by the egress router pod. Optional: You can include the subnet length, the **/24** suffix, so that a proper route to the local subnet is set. If you do not specify a subnet length, then the egress router can access only the host specified with the **EGRESS_GATEWAY** variable and no other hosts on the subnet.
3	Same value as the default gateway used by the node.
4	Specify a list of one or more proxy destinations.
5	Optional: Specify to output the DNS proxy log output to **stdout**.

28.12.2. Egress destination configuration format

When the router is deployed in DNS proxy mode, you specify a list of port and destination mappings. A destination may be either an IP address or a DNS name.

An egress router pod supports the following formats for specifying port and destination mappings:

Port and remote address

> You can specify a source port and a destination host by using the two field format: **<port> <remote_address>**.

The host can be an IP address or a DNS name. If a DNS name is provided, DNS resolution occurs at runtime. For a given host, the proxy connects to the specified source port on the destination host when connecting to the destination host IP address.

Port and remote address pair example

```
80 172.16.12.11
100 example.com
```

Port, remote address, and remote port

> You can specify a source port, a destination host, and a destination port by using the three field
> format: **<port> <remote_address> <remote_port>**.

The three field format behaves identically to the two field version, with the exception that the
destination port can be different than the source port.

Port, remote address, and remote port example

```
8080 192.168.60.252 80
8443 web.example.com 443
```

28.12.3. Deploying an egress router pod in DNS proxy mode

In *DNS proxy mode* , an egress router pod acts as a DNS proxy for TCP-based services from its own IP
address to one or more destination IP addresses.

Prerequisites

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

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

Procedure

1. Create an egress router pod.

2. Create a service for the egress router pod:

   a. Create a file named **egress-router-service.yaml** that contains the following YAML. Set
      spec.ports to the list of ports that you defined previously for the
      EGRESS_DNS_PROXY_DESTINATION environment variable.

      ```yaml
      apiVersion: v1
      kind: Service
      metadata:
        name: egress-dns-svc
      spec:
        ports:
          ...
        type: ClusterIP
        selector:
          name: egress-dns-proxy
      ```

      For example:

      ```yaml
      apiVersion: v1
      kind: Service
      metadata:
        name: egress-dns-svc
      spec:
      ```

```
      ports:
      - name: con1
        protocol: TCP
        port: 80
        targetPort: 80
      - name: con2
        protocol: TCP
        port: 100
        targetPort: 100
      type: ClusterIP
      selector:
        name: egress-dns-proxy
```

b. To create the service, enter the following command:

```
$ oc create -f egress-router-service.yaml
```

Pods can now connect to this service. The connections are proxied to the corresponding ports on the external server, using the reserved egress IP address.

28.12.4. Additional resources

- Configuring an egress router destination mappings with a ConfigMap

28.13. CONFIGURING AN EGRESS ROUTER POD DESTINATION LIST FROM A CONFIG MAP

As a cluster administrator, you can define a **ConfigMap** object that specifies destination mappings for an egress router pod. The specific format of the configuration depends on the type of egress router pod. For details on the format, refer to the documentation for the specific egress router pod.

28.13.1. Configuring an egress router destination mappings with a config map

For a large or frequently-changing set of destination mappings, you can use a config map to externally maintain the list. An advantage of this approach is that permission to edit the config map can be delegated to users without **cluster-admin** privileges. Because the egress router pod requires a privileged container, it is not possible for users without **cluster-admin** privileges to edit the pod definition directly.

> **NOTE**
>
> The egress router pod does not automatically update when the config map changes. You must restart the egress router pod to get updates.

Prerequisites

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

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

Procedure

1. Create a file containing the mapping data for the egress router pod, as in the following example:

```
# Egress routes for Project "Test", version 3

80   tcp 203.0.113.25

8080 tcp 203.0.113.26 80
8443 tcp 203.0.113.26 443

# Fallback
203.0.113.27
```

You can put blank lines and comments into this file.

2. Create a **ConfigMap** object from the file:

```
$ oc delete configmap egress-routes --ignore-not-found
```

```
$ oc create configmap egress-routes \
  --from-file=destination=my-egress-destination.txt
```

In the previous command, the **egress-routes** value is the name of the **ConfigMap** object to create and **my-egress-destination.txt** is the name of the file that the data is read from.

TIP

You can alternatively apply the following YAML to create the config map:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: egress-routes
data:
  destination: |
    # Egress routes for Project "Test", version 3

    80   tcp 203.0.113.25

    8080 tcp 203.0.113.26 80
    8443 tcp 203.0.113.26 443

    # Fallback
    203.0.113.27
```

3. Create an egress router pod definition and specify the **configMapKeyRef** stanza for the **EGRESS_DESTINATION** field in the environment stanza:

```
...
env:
- name: EGRESS_DESTINATION
  valueFrom:
    configMapKeyRef:
      name: egress-routes
      key: destination
...
```

28.13.2. Additional resources

- [Redirect mode](#)

- [HTTP proxy mode](#)

- [DNS proxy mode](#)

28.14. ENABLING MULTICAST FOR A PROJECT

28.14.1. About multicast

With IP multicast, data is broadcast to many IP addresses simultaneously.

IMPORTANT

- At this time, multicast is best used for low-bandwidth coordination or service discovery and not a high-bandwidth solution.

- By default, network policies affect all connections in a namespace. However, multicast is unaffected by network policies. If multicast is enabled in the same namespace as your network policies, it is always allowed, even if there is a **deny-all** network policy. Cluster administrators should consider the implications to the exemption of multicast from network policies before enabling it.

Multicast traffic between OpenShift Container Platform pods is disabled by default. If you are using the OpenShift SDN network plugin, you can enable multicast on a per-project basis.

When using the OpenShift SDN network plugin in **networkpolicy** isolation mode:

- Multicast packets sent by a pod will be delivered to all other pods in the project, regardless of **NetworkPolicy** objects. Pods might be able to communicate over multicast even when they cannot communicate over unicast.

- Multicast packets sent by a pod in one project will never be delivered to pods in any other project, even if there are **NetworkPolicy** objects that allow communication between the projects.

When using the OpenShift SDN network plugin in **multitenant** isolation mode:

- Multicast packets sent by a pod will be delivered to all other pods in the project.

- Multicast packets sent by a pod in one project will be delivered to pods in other projects only if each project is joined together and multicast is enabled in each joined project.

28.14.2. Enabling multicast between pods

You can enable multicast between pods for your project.

Prerequisites

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

- You must log in to the cluster with a user that has the **cluster-admin** role.

Procedure

- Run the following command to enable multicast for a project. Replace **\<namespace>** with the namespace for the project you want to enable multicast for.

```
$ oc annotate netnamespace <namespace> \
    netnamespace.network.openshift.io/multicast-enabled=true
```

Verification

To verify that multicast is enabled for a project, complete the following procedure:

1. Change your current project to the project that you enabled multicast for. Replace **\<project>** with the project name.

   ```
   $ oc project <project>
   ```

2. Create a pod to act as a multicast receiver:

   ```
   $ cat <<EOF| oc create -f -
   apiVersion: v1
   kind: Pod
   metadata:
     name: mlistener
     labels:
       app: multicast-verify
   spec:
     containers:
       - name: mlistener
         image: registry.access.redhat.com/ubi9
         command: ["/bin/sh", "-c"]
         args:
           ["dnf -y install socat hostname && sleep inf"]
         ports:
           - containerPort: 30102
             name: mlistener
             protocol: UDP
   EOF
   ```

3. Create a pod to act as a multicast sender:

   ```
   $ cat <<EOF| oc create -f -
   apiVersion: v1
   kind: Pod
   metadata:
     name: msender
     labels:
       app: multicast-verify
   spec:
     containers:
       - name: msender
         image: registry.access.redhat.com/ubi9
         command: ["/bin/sh", "-c"]
   ```

```
    args:
      ["dnf -y install socat && sleep inf"]
EOF
```

4. In a new terminal window or tab, start the multicast listener.

   a. Get the IP address for the Pod:

   ```
   $ POD_IP=$(oc get pods mlistener -o jsonpath='{.status.podIP}')
   ```

   b. Start the multicast listener by entering the following command:

   ```
   $ oc exec mlistener -i -t -- \
       socat UDP4-RECVFROM:30102,ip-add-membership=224.1.0.1:$POD_IP,fork
   EXEC:hostname
   ```

5. Start the multicast transmitter.

   a. Get the pod network IP address range:

   ```
   $ CIDR=$(oc get Network.config.openshift.io cluster \
       -o jsonpath='{.status.clusterNetwork[0].cidr}')
   ```

   b. To send a multicast message, enter the following command:

   ```
   $ oc exec msender -i -t -- \
       /bin/bash -c "echo | socat STDIO UDP4-
   DATAGRAM:224.1.0.1:30102,range=$CIDR,ip-multicast-ttl=64"
   ```

   If multicast is working, the previous command returns the following output:

   ```
   mlistener
   ```

28.15. DISABLING MULTICAST FOR A PROJECT

28.15.1. Disabling multicast between pods

You can disable multicast between pods for your project.

Prerequisites

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

- You must log in to the cluster with a user that has the **cluster-admin** role.

Procedure

- Disable multicast by running the following command:

  ```
  $ oc annotate netnamespace <namespace> \ 1
      netnamespace.network.openshift.io/multicast-enabled-
  ```

1 The **namespace** for the project you want to disable multicast for.

28.16. CONFIGURING NETWORK ISOLATION USING OPENSHIFT SDN

When your cluster is configured to use the multitenant isolation mode for the OpenShift SDN network plugin, each project is isolated by default. Network traffic is not allowed between pods or services in different projects in multitenant isolation mode.

You can change the behavior of multitenant isolation for a project in two ways:

- You can join one or more projects, allowing network traffic between pods and services in different projects.

- You can disable network isolation for a project. It will be globally accessible, accepting network traffic from pods and services in all other projects. A globally accessible project can access pods and services in all other projects.

28.16.1. Prerequisites

- You must have a cluster configured to use the OpenShift SDN network plugin in multitenant isolation mode.

28.16.2. Joining projects

You can join two or more projects to allow network traffic between pods and services in different projects.

Prerequisites

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

- You must log in to the cluster with a user that has the **cluster-admin** role.

Procedure

1. Use the following command to join projects to an existing project network:

   ```
   $ oc adm pod-network join-projects --to=<project1> <project2> <project3>
   ```

   Alternatively, instead of specifying specific project names, you can use the **--selector=<project_selector>** option to specify projects based upon an associated label.

2. Optional: Run the following command to view the pod networks that you have joined together:

   ```
   $ oc get netnamespaces
   ```

   Projects in the same pod-network have the same network ID in the **NETID** column.

28.16.3. Isolating a project

You can isolate a project so that pods and services in other projects cannot access its pods and services.

Prerequisites

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

- You must log in to the cluster with a user that has the **cluster-admin** role.

Procedure

- To isolate the projects in the cluster, run the following command:

  ```
  $ oc adm pod-network isolate-projects <project1> <project2>
  ```

  Alternatively, instead of specifying specific project names, you can use the **--selector=<project_selector>** option to specify projects based upon an associated label.

28.16.4. Disabling network isolation for a project

You can disable network isolation for a project.

Prerequisites

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

- You must log in to the cluster with a user that has the **cluster-admin** role.

Procedure

- Run the following command for the project:

  ```
  $ oc adm pod-network make-projects-global <project1> <project2>
  ```

  Alternatively, instead of specifying specific project names, you can use the **--selector=<project_selector>** option to specify projects based upon an associated label.

28.17. CONFIGURING KUBE-PROXY

The Kubernetes network proxy (kube-proxy) runs on each node and is managed by the Cluster Network Operator (CNO). kube-proxy maintains network rules for forwarding connections for endpoints associated with services.

28.17.1. About iptables rules synchronization

The synchronization period determines how frequently the Kubernetes network proxy (kube-proxy) syncs the iptables rules on a node.

A sync begins when either of the following events occurs:

- An event occurs, such as service or endpoint is added to or removed from the cluster.

- The time since the last sync exceeds the sync period defined for kube-proxy.

28.17.2. kube-proxy configuration parameters

You can modify the following **kubeProxyConfig** parameters.

> **NOTE**
>
> Because of performance improvements introduced in OpenShift Container Platform 4.3 and greater, adjusting the **iptablesSyncPeriod** parameter is no longer necessary.

Table 28.3. Parameters

Parameter	Description	Values	Default
iptablesSyncPeriod	The refresh period for **iptables** rules.	A time interval, such as **30s** or **2m**. Valid suffixes include **s**, **m**, and **h** and are described in the Go time package documentation.	**30s**
proxyArguments.iptables-min-sync-period	The minimum duration before refreshing **iptables** rules. This parameter ensures that the refresh does not happen too frequently. By default, a refresh starts as soon as a change that affects **iptables** rules occurs.	A time interval, such as **30s** or **2m**. Valid suffixes include **s**, **m**, and **h** and are described in the Go time package	**0s**

28.17.3. Modifying the kube-proxy configuration

You can modify the Kubernetes network proxy configuration for your cluster.

Prerequisites

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

- Log in to a running cluster with the **cluster-admin** role.

Procedure

1. Edit the **Network.operator.openshift.io** custom resource (CR) by running the following command:

   ```
   $ oc edit network.operator.openshift.io cluster
   ```

2. Modify the **kubeProxyConfig** parameter in the CR with your changes to the kube-proxy configuration, such as in the following example CR:

   ```
   apiVersion: operator.openshift.io/v1
   kind: Network
   metadata:
     name: cluster
   spec:
     kubeProxyConfig:
   ```

```
iptablesSyncPeriod: 30s
proxyArguments:
  iptables-min-sync-period: ["30s"]
```

3. Save the file and exit the text editor.
   The syntax is validated by the **oc** command when you save the file and exit the editor. If your modifications contain a syntax error, the editor opens the file and displays an error message.

4. Enter the following command to confirm the configuration update:

```
$ oc get networks.operator.openshift.io -o yaml
```

Example output

```
apiVersion: v1
items:
- apiVersion: operator.openshift.io/v1
  kind: Network
  metadata:
    name: cluster
  spec:
    clusterNetwork:
    - cidr: 10.128.0.0/14
      hostPrefix: 23
    defaultNetwork:
      type: OpenShiftSDN
    kubeProxyConfig:
      iptablesSyncPeriod: 30s
      proxyArguments:
        iptables-min-sync-period:
        - 30s
    serviceNetwork:
    - 172.30.0.0/16
  status: {}
kind: List
```

5. Optional: Enter the following command to confirm that the Cluster Network Operator accepted the configuration change:

```
$ oc get clusteroperator network
```

Example output

```
NAME     VERSION   AVAILABLE  PROGRESSING  DEGRADED  SINCE
network  4.1.0-0.9  True       False        False     1m
```

The **AVAILABLE** field is **True** when the configuration update is applied successfully.

CHAPTER 29. CONFIGURING ROUTES

29.1. ROUTE CONFIGURATION

29.1.1. Creating an HTTP-based route

A route allows you to host your application at a public URL. It can either be secure or unsecured, depending on the network security configuration of your application. An HTTP-based route is an unsecured route that uses the basic HTTP routing protocol and exposes a service on an unsecured application port.

The following procedure describes how to create a simple HTTP-based route to a web application, using the **hello-openshift** application as an example.

Prerequisites

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

- You are logged in as an administrator.

- You have a web application that exposes a port and a TCP endpoint listening for traffic on the port.

Procedure

1. Create a project called **hello-openshift** by running the following command:

   ```
   $ oc new-project hello-openshift
   ```

2. Create a pod in the project by running the following command:

   ```
   $ oc create -f https://raw.githubusercontent.com/openshift/origin/master/examples/hello-openshift/hello-pod.json
   ```

3. Create a service called **hello-openshift** by running the following command:

   ```
   $ oc expose pod/hello-openshift
   ```

4. Create an unsecured route to the **hello-openshift** application by running the following command:

   ```
   $ oc expose svc hello-openshift
   ```

Verification

- To verify that the **route** resource that you created, run the following command:

   ```
   $ oc get routes -o yaml <name of resource>   1
   ```

   1 In this example, the route is named **hello-openshift**.

Sample YAML definition of the created unsecured route:

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: hello-openshift
spec:
  host: hello-openshift-hello-openshift.<Ingress_Domain>    1
  port:
    targetPort: 8080    2
  to:
    kind: Service
    name: hello-openshift
```

1   **<Ingress_Domain>** is the default ingress domain name. The **ingresses.config/cluster** object is created during the installation and cannot be changed. If you want to specify a different domain, you can specify an alternative cluster domain using the **appsDomain** option.

2   **targetPort** is the target port on pods that is selected by the service that this route points to.

> **NOTE**
>
> To display your default ingress domain, run the following command:
>
> ```
> $ oc get ingresses.config/cluster -o jsonpath={.spec.domain}
> ```

29.1.2. Creating a route for Ingress Controller sharding

A route allows you to host your application at a URL. In this case, the hostname is not set and the route uses a subdomain instead. When you specify a subdomain, you automatically use the domain of the Ingress Controller that exposes the route. For situations where a route is exposed by multiple Ingress Controllers, the route is hosted at multiple URLs.

The following procedure describes how to create a route for Ingress Controller sharding, using the **hello-openshift** application as an example.

Ingress Controller sharding is useful when balancing incoming traffic load among a set of Ingress Controllers and when isolating traffic to a specific Ingress Controller. For example, company A goes to one Ingress Controller and company B to another.

Prerequisites

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

- You are logged in as a project administrator.

- You have a web application that exposes a port and an HTTP or TLS endpoint listening for traffic on the port.

- You have configured the Ingress Controller for sharding.

Procedure

1. Create a project called **hello-openshift** by running the following command:

   ```
   $ oc new-project hello-openshift
   ```

2. Create a pod in the project by running the following command:

   ```
   $ oc create -f https://raw.githubusercontent.com/openshift/origin/master/examples/hello-openshift/hello-pod.json
   ```

3. Create a service called **hello-openshift** by running the following command:

   ```
   $ oc expose pod/hello-openshift
   ```

4. Create a route definition called **hello-openshift-route.yaml**:

   YAML definition of the created route for sharding:

   ```
   apiVersion: route.openshift.io/v1
   kind: Route
   metadata:
    labels:
     type: sharded ❶
    name: hello-openshift-edge
    namespace: hello-openshift
   spec:
    subdomain: hello-openshift ❷
    tls:
     termination: edge
    to:
     kind: Service
     name: hello-openshift
   ```

   ❶ Both the label key and its corresponding label value must match the ones specified in the Ingress Controller. In this example, the Ingress Controller has the label key and value **type: sharded**.

   ❷ The route will be exposed using the value of the **subdomain** field. When you specify the **subdomain** field, you must leave the hostname unset. If you specify both the **host** and **subdomain** fields, then the route will use the value of the **host** field, and ignore the **subdomain** field.

5. Use **hello-openshift-route.yaml** to create a route to the **hello-openshift** application by running the following command:

   ```
   $ oc -n hello-openshift create -f hello-openshift-route.yaml
   ```

Verification

- Get the status of the route with the following command:

  ```
  $ oc -n hello-openshift get routes/hello-openshift-edge -o yaml
  ```

The resulting **Route** resource should look similar to the following:

Example output

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  labels:
    type: sharded
  name: hello-openshift-edge
  namespace: hello-openshift
spec:
  subdomain: hello-openshift
  tls:
    termination: edge
  to:
    kind: Service
    name: hello-openshift
status:
  ingress:
  - host: hello-openshift.<apps-sharded.basedomain.example.net> 1
    routerCanonicalHostname: router-sharded.<apps-sharded.basedomain.example.net> 2
    routerName: sharded 3
```

1 The hostname the Ingress Controller, or router, uses to expose the route. The value of the **host** field is automatically determined by the Ingress Controller, and uses its domain. In this example, the domain of the Ingress Controller is **<apps-sharded.basedomain.example.net>**.

2 The hostname of the Ingress Controller.

3 The name of the Ingress Controller. In this example, the Ingress Controller has the name **sharded**.

29.1.3. Configuring route timeouts

You can configure the default timeouts for an existing route when you have services in need of a low timeout, which is required for Service Level Availability (SLA) purposes, or a high timeout, for cases with a slow back end.

Prerequisites

- You need a deployed Ingress Controller on a running cluster.

Procedure

1. Using the **oc annotate** command, add the timeout to the route:

   ```
   $ oc annotate route <route_name> \
       --overwrite haproxy.router.openshift.io/timeout=<timeout><time_unit> 1
   ```

   1 Supported time units are microseconds (us), milliseconds (ms), seconds (s), minutes (m), hours (h), or days (d).

The following example sets a timeout of two seconds on a route named **myroute**:

```
$ oc annotate route myroute --overwrite haproxy.router.openshift.io/timeout=2s
```

29.1.4. HTTP Strict Transport Security

HTTP Strict Transport Security (HSTS) policy is a security enhancement, which signals to the browser client that only HTTPS traffic is allowed on the route host. HSTS also optimizes web traffic by signaling HTTPS transport is required, without using HTTP redirects. HSTS is useful for speeding up interactions with websites.

When HSTS policy is enforced, HSTS adds a Strict Transport Security header to HTTP and HTTPS responses from the site. You can use the **insecureEdgeTerminationPolicy** value in a route to redirect HTTP to HTTPS. When HSTS is enforced, the client changes all requests from the HTTP URL to HTTPS before the request is sent, eliminating the need for a redirect.

Cluster administrators can configure HSTS to do the following:

- Enable HSTS per-route

- Disable HSTS per-route

- Enforce HSTS per-domain, for a set of domains, or use namespace labels in combination with domains
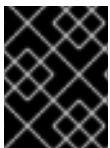
> **IMPORTANT**
>
> HSTS works only with secure routes, either edge-terminated or re-encrypt. The configuration is ineffective on HTTP or passthrough routes.

29.1.4.1. Enabling HTTP Strict Transport Security per-route

HTTP strict transport security (HSTS) is implemented in the HAProxy template and applied to edge and re-encrypt routes that have the **haproxy.router.openshift.io/hsts_header** annotation.

Prerequisites

- You are logged in to the cluster with a user with administrator privileges for the project.

- You installed the **oc** CLI.

Procedure

- To enable HSTS on a route, add the **haproxy.router.openshift.io/hsts_header** value to the edge-terminated or re-encrypt route. You can use the **oc annotate** tool to do this by running the following command:

```
$ oc annotate route <route_name> -n <namespace> --overwrite=true
"haproxy.router.openshift.io/hsts_header"="max-age=31536000;\  1
includeSubDomains;preload"
```

    1 In this example, the maximum age is set to **31536000** ms, which is approximately eight and a half hours.

> **NOTE**
>
> In this example, the equal sign (**=**) is in quotes. This is required to properly execute the annotate command.

Example route configured with an annotation

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  annotations:
    haproxy.router.openshift.io/hsts_header: max-age=31536000;includeSubDomains;preload
    ① ② ③
...
spec:
  host: def.abc.com
  tls:
    termination: "reencrypt"
    ...
  wildcardPolicy: "Subdomain"
```

① Required. **max-age** measures the length of time, in seconds, that the HSTS policy is in effect. If set to **0**, it negates the policy.

② Optional. When included, **includeSubDomains** tells the client that all subdomains of the host must have the same HSTS policy as the host.

③ Optional. When **max-age** is greater than 0, you can add **preload** in **haproxy.router.openshift.io/hsts_header** to allow external services to include this site in their HSTS preload lists. For example, sites such as Google can construct a list of sites that have **preload** set. Browsers can then use these lists to determine which sites they can communicate with over HTTPS, even before they have interacted with the site. Without **preload** set, browsers must have interacted with the site over HTTPS, at least once, to get the header.

29.1.4.2. Disabling HTTP Strict Transport Security per-route

To disable HTTP strict transport security (HSTS) per-route, you can set the **max-age** value in the route annotation to **0**.

Prerequisites

- You are logged in to the cluster with a user with administrator privileges for the project.

- You installed the **oc** CLI.

Procedure

- To disable HSTS, set the **max-age** value in the route annotation to **0**, by entering the following command:

```
$ oc annotate route <route_name> -n <namespace> --overwrite=true
"haproxy.router.openshift.io/hsts_header"="max-age=0"
```

TIP

You can alternatively apply the following YAML to create the config map:

Example of disabling HSTS per-route

```
metadata:
  annotations:
    haproxy.router.openshift.io/hsts_header: max-age=0
```

- To disable HSTS for every route in a namespace, enter the following command:

```
$ oc annotate route --all -n <namespace> --overwrite=true
"haproxy.router.openshift.io/hsts_header"="max-age=0"
```

Verification

1. To query the annotation for all routes, enter the following command:

```
$ oc get route  --all-namespaces -o go-template='{{range .items}}{{if .metadata.annotations}}
{{$a := index .metadata.annotations "haproxy.router.openshift.io/hsts_header"}}{{$n :=
.metadata.name}}{{with $a}}Name: {{$n}} HSTS: {{$a}}{{"\n"}}{{else}}{{""}}{{end}}{{end}}
{{end}}'
```

Example output

```
Name: routename HSTS: max-age=0
```

29.1.4.3. Enforcing HTTP Strict Transport Security per-domain

To enforce HTTP Strict Transport Security (HSTS) per-domain for secure routes, add a
requiredHSTSPolicies record to the Ingress spec to capture the configuration of the HSTS policy.

If you configure a **requiredHSTSPolicy** to enforce HSTS, then any newly created route must be
configured with a compliant HSTS policy annotation.

> **NOTE**
>
> To handle upgraded clusters with non-compliant HSTS routes, you can update the
> manifests at the source and apply the updates.

> **NOTE**
>
> You cannot use **oc expose route** or **oc create route** commands to add a route in a
> domain that enforces HSTS, because the API for these commands does not accept
> annotations.

> **IMPORTANT**
>
> HSTS cannot be applied to insecure, or non-TLS routes, even if HSTS is requested for all
> routes globally.

Prerequisites

- You are logged in to the cluster with a user with administrator privileges for the project.

- You installed the **oc** CLI.

Procedure

1. Edit the Ingress config file:

   ```
   $ oc edit ingresses.config.openshift.io/cluster
   ```

Example HSTS policy

```
apiVersion: config.openshift.io/v1
kind: Ingress
metadata:
  name: cluster
spec:
  domain: 'hello-openshift-default.apps.username.devcluster.openshift.com'
  requiredHSTSPolicies: 1
  - domainPatterns: 2
    - '*hello-openshift-default.apps.username.devcluster.openshift.com'
    - '*hello-openshift-default2.apps.username.devcluster.openshift.com'
    namespaceSelector: 3
      matchLabels:
        myPolicy: strict
    maxAge: 4
      smallestMaxAge: 1
      largestMaxAge: 31536000
    preloadPolicy: RequirePreload 5
    includeSubDomainsPolicy: RequireIncludeSubDomains 6
  - domainPatterns: 7
    - 'abc.example.com'
    - '*xyz.example.com'
    namespaceSelector:
      matchLabels: {}
    maxAge: {}
    preloadPolicy: NoOpinion
    includeSubDomainsPolicy: RequireNoIncludeSubDomains
```

1   Required. **requiredHSTSPolicies** are validated in order, and the first matching **domainPatterns** applies.

2 **7** Required. You must specify at least one **domainPatterns** hostname. Any number of domains can be listed. You can include multiple sections of enforcing options for different **domainPatterns**.

3   Optional. If you include **namespaceSelector**, it must match the labels of the project where the routes reside, to enforce the set HSTS policy on the routes. Routes that only match the **namespaceSelector** and not the **domainPatterns** are not validated.

4   Required. **max-age** measures the length of time, in seconds, that the HSTS policy is in effect. This policy setting allows for a smallest and largest **max-age** to be enforced.

- The **largestMaxAge** value must be between **0** and **2147483647**. It can be left unspecified, which means no upper limit is enforced.

- The **smallestMaxAge** value must be between **0** and **2147483647**. Enter **0** to disable HSTS for troubleshooting, otherwise enter **1** if you never want HSTS to be disabled. It can be left unspecified, which means no lower limit is enforced.

5 Optional. Including **preload** in **haproxy.router.openshift.io/hsts_header** allows external services to include this site in their HSTS preload lists. Browsers can then use these lists to determine which sites they can communicate with over HTTPS, before they have interacted with the site. Without **preload** set, browsers need to interact at least once with the site to get the header. **preload** can be set with one of the following:

- **RequirePreload**: **preload** is required by the **RequiredHSTSPolicy**.

- **RequireNoPreload**: **preload** is forbidden by the **RequiredHSTSPolicy**.

- **NoOpinion**: **preload** does not matter to the **RequiredHSTSPolicy**.

6 Optional. **includeSubDomainsPolicy** can be set with one of the following:

- **RequireIncludeSubDomains**: **includeSubDomains** is required by the **RequiredHSTSPolicy**.

- **RequireNoIncludeSubDomains**: **includeSubDomains** is forbidden by the **RequiredHSTSPolicy**.

- **NoOpinion**: **includeSubDomains** does not matter to the **RequiredHSTSPolicy**.

2. You can apply HSTS to all routes in the cluster or in a particular namespace by entering the **oc annotate command**.

   - To apply HSTS to all routes in the cluster, enter the **oc annotate command**. For example:

     ```
     $ oc annotate route --all --all-namespaces --overwrite=true
     "haproxy.router.openshift.io/hsts_header"="max-age=31536000"
     ```

   - To apply HSTS to all routes in a particular namespace, enter the **oc annotate command**. For example:

     ```
     $ oc annotate route --all -n my-namespace --overwrite=true
     "haproxy.router.openshift.io/hsts_header"="max-age=31536000"
     ```

Verification

You can review the HSTS policy you configured. For example:

- To review the **maxAge** set for required HSTS policies, enter the following command:

  ```
  $ oc get clusteroperator/ingress -n openshift-ingress-operator -o jsonpath='{range
  .spec.requiredHSTSPolicies[*]}{.spec.requiredHSTSPolicies.maxAgePolicy.largestMaxAge}
  {"\n"}{end}'
  ```

- To review the HSTS annotations on all routes, enter the following command:

```
$ oc get route  --all-namespaces -o go-template='{{range .items}}{{if .metadata.annotations}}
{{$a := index .metadata.annotations "haproxy.router.openshift.io/hsts_header"}}{{$n :=
.metadata.name}}{{with $a}}Name: {{$n}} HSTS: {{$a}}{{"\n"}}{{else}}{{""}}{{end}}{{end}}
{{end}}'
```

Example output

```
Name: <_routename_> HSTS: max-age=31536000;preload;includeSubDomains
```

29.1.5. Throughput issue troubleshooting methods

Sometimes applications deployed by using OpenShift Container Platform can cause network throughput issues, such as unusually high latency between specific services.

If pod logs do not reveal any cause of the problem, use the following methods to analyze performance issues:

- Use a packet analyzer, such as **ping** or **tcpdump** to analyze traffic between a pod and its node. For example, run the **tcpdump** tool on each pod while reproducing the behavior that led to the issue. Review the captures on both sides to compare send and receive timestamps to analyze the latency of traffic to and from a pod. Latency can occur in OpenShift Container Platform if a node interface is overloaded with traffic from other pods, storage devices, or the data plane.

  ```
  $ tcpdump -s 0 -i any -w /tmp/dump.pcap host <podip 1> && host <podip 2>  ❶
  ```

  ❶   **podip** is the IP address for the pod. Run the **oc get pod <pod_name> -o wide** command to get the IP address of a pod.

  The **tcpdump** command generates a file at **/tmp/dump.pcap** containing all traffic between these two pods. You can run the analyzer shortly before the issue is reproduced and stop the analyzer shortly after the issue is finished reproducing to minimize the size of the file. You can also run a packet analyzer between the nodes (eliminating the SDN from the equation) with:

  ```
  $ tcpdump -s 0 -i any -w /tmp/dump.pcap port 4789
  ```

- Use a bandwidth measuring tool, such as **iperf**, to measure streaming throughput and UDP throughput. Locate any bottlenecks by running the tool from the pods first, and then running it from the nodes.

  - For information on installing and using **iperf**, see this Red Hat Solution.

- In some cases, the cluster may mark the node with the router pod as unhealthy due to latency issues. Use worker latency profiles to adjust the frequency that the cluster waits for a status update from the node before taking action.

- If your cluster has designated lower-latency and higher-latency nodes, configure the **spec.nodePlacement** field in the Ingress Controller to control the placement of the router pod.

Additional resources

- Latency spikes or temporary reduction in throughput to remote workers

- Ingress Controller configuration parameters

29.1.6. Using cookies to keep route statefulness

OpenShift Container Platform provides sticky sessions, which enables stateful application traffic by ensuring all traffic hits the same endpoint. However, if the endpoint pod terminates, whether through restart, scaling, or a change in configuration, this statefulness can disappear.

OpenShift Container Platform can use cookies to configure session persistence. The Ingress controller selects an endpoint to handle any user requests, and creates a cookie for the session. The cookie is passed back in the response to the request and the user sends the cookie back with the next request in the session. The cookie tells the Ingress Controller which endpoint is handling the session, ensuring that client requests use the cookie so that they are routed to the same pod.
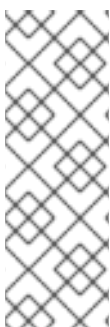
> **NOTE**
>
> Cookies cannot be set on passthrough routes, because the HTTP traffic cannot be seen. Instead, a number is calculated based on the source IP address, which determines the backend.
>
> If backends change, the traffic can be directed to the wrong server, making it less sticky. If you are using a load balancer, which hides source IP, the same number is set for all connections and traffic is sent to the same pod.

29.1.6.1. Annotating a route with a cookie

You can set a cookie name to overwrite the default, auto-generated one for the route. This allows the application receiving route traffic to know the cookie name. By deleting the cookie it can force the next request to re-choose an endpoint. So, if a server was overloaded it tries to remove the requests from the client and redistribute them.

Procedure

1. Annotate the route with the specified cookie name:

   ```
   $ oc annotate route <route_name> router.openshift.io/cookie_name="<cookie_name>"
   ```

   where:

   <route_name>

   Specifies the name of the route.

   <cookie_name>

   Specifies the name for the cookie.

   For example, to annotate the route **my_route** with the cookie name **my_cookie**:

   ```
   $ oc annotate route my_route router.openshift.io/cookie_name="my_cookie"
   ```

2. Capture the route hostname in a variable:

   ```
   $ ROUTE_NAME=$(oc get route <route_name> -o jsonpath='{.spec.host}')
   ```

   where:

   <route_name>

Specifies the name of the route.

3. Save the cookie, and then access the route:

```
$ curl $ROUTE_NAME -k -c /tmp/cookie_jar
```

Use the cookie saved by the previous command when connecting to the route:

```
$ curl $ROUTE_NAME -k -b /tmp/cookie_jar
```

29.1.7. Path-based routes

Path-based routes specify a path component that can be compared against a URL, which requires that the traffic for the route be HTTP based. Thus, multiple routes can be served using the same hostname, each with a different path. Routers should match routes based on the most specific path to the least.

The following table shows example routes and their accessibility:

Table 29.1. Route availability

Route	When Compared to	Accessible
www.example.com/test	*www.example.com/test*	Yes
	www.example.com	No
www.example.com/test and *www.example.com*	*www.example.com/test*	Yes
	www.example.com	Yes
www.example.com	*www.example.com/text*	Yes (Matched by the host, not the route)
	www.example.com	Yes

An unsecured route with a path

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: route-unsecured
spec:
  host: www.example.com
  path: "/test" 1
  to:
    kind: Service
    name: service-name
```

1   The path is the only added attribute for a path-based route.

> **NOTE**
>
> Path-based routing is not available when using passthrough TLS, as the router does not terminate TLS in that case and cannot read the contents of the request.

29.1.8. Route-specific annotations

The Ingress Controller can set the default options for all the routes it exposes. An individual route can override some of these defaults by providing specific configurations in its annotations. Red Hat does not support adding a route annotation to an operator-managed route.
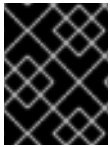
> **IMPORTANT**
>
> To create a whitelist with multiple source IPs or subnets, use a space-delimited list. Any other delimiter type causes the list to be ignored without a warning or error message.

Table 29.2. Route annotations

Variable	Description	Environment variable used as default
haproxy.router.openshift.io/balance	Sets the load-balancing algorithm. Available options are **random**, **source**, **roundrobin**, and **leastconn**. The default value is **source** for TLS passthrough routes. For all other routes, the default is **random**.	**ROUTER_TCP_BALANCE_SCHEME** for passthrough routes. Otherwise, use **ROUTER_LOAD_BALANCE_ALGORITHM**.
haproxy.router.openshift.io/disable_cookies	Disables the use of cookies to track related connections. If set to **'true'** or **'TRUE'**, the balance algorithm is used to choose which back-end serves connections for each incoming HTTP request.	
router.openshift.io/cookie_name	Specifies an optional cookie to use for this route. The name must consist of any combination of upper and lower case letters, digits, "_", and "-". The default is the hashed internal key name for the route.	

Variable	Description	Environment variable used as default
haproxy.router.openshift.io/pod-concurrent-connections	Sets the maximum number of connections that are allowed to a backing pod from a router. Note: If there are multiple pods, each can have this many connections. If you have multiple routers, there is no coordination among them, each may connect this many times. If not set, or set to 0, there is no limit.	
haproxy.router.openshift.io/rate-limit-connections	Setting **'true'** or **'TRUE'** enables rate limiting functionality which is implemented through stick-tables on the specific backend per route. Note: Using this annotation provides basic protection against denial-of-service attacks.	
haproxy.router.openshift.io/rate-limit-connections.concurrent-tcp	Limits the number of concurrent TCP connections made through the same source IP address. It accepts a numeric value. Note: Using this annotation provides basic protection against denial-of-service attacks.	
haproxy.router.openshift.io/rate-limit-connections.rate-http	Limits the rate at which a client with the same source IP address can make HTTP requests. It accepts a numeric value. Note: Using this annotation provides basic protection against denial-of-service attacks.	
haproxy.router.openshift.io/rate-limit-connections.rate-tcp	Limits the rate at which a client with the same source IP address can make TCP connections. It accepts a numeric value. Note: Using this annotation provides basic protection against denial-of-service attacks.	
haproxy.router.openshift.io/timeout	Sets a server-side timeout for the route. (TimeUnits)	**ROUTER_DEFAULT_SERVER_TIMEOUT**

Variable	Description	Environment variable used as default
haproxy.router.openshift.io/timeout-tunnel	This timeout applies to a tunnel connection, for example, WebSocket over cleartext, edge, reencrypt, or passthrough routes. With cleartext, edge, or reencrypt route types, this annotation is applied as a timeout tunnel with the existing timeout value. For the passthrough route types, the annotation takes precedence over any existing timeout value set.	**ROUTER_DEFAULT_TUNNEL_TIMEOUT**
ingresses.config/cluster ingress.operator.openshift.io/hard-stop-after	You can set either an IngressController or the ingress config . This annotation redeploys the router and configures the HA proxy to emit the haproxy **hard-stop-after** global option, which defines the maximum time allowed to perform a clean soft-stop.	**ROUTER_HARD_STOP_AFTER**
router.openshift.io/haproxy.health.check.interval	Sets the interval for the back-end health checks. (TimeUnits)	**ROUTER_BACKEND_CHECK_INTERVAL**
haproxy.router.openshift.io/ip_whitelist	Sets an allowlist for the route. The allowlist is a space-separated list of IP addresses and CIDR ranges for the approved source addresses. Requests from IP addresses that are not in the allowlist are dropped. The maximum number of IP addresses and CIDR ranges directly visible in the **haproxy.config** file is 61. [1]	
haproxy.router.openshift.io/hsts_header	Sets a Strict-Transport-Security header for the edge terminated or re-encrypt route.	
haproxy.router.openshift.io/rewrite-target	Sets the rewrite path of the request on the backend.	

Variable	Description	Environment variable used as default
router.openshift.io/cookie-same-site	Sets a value to restrict cookies. The values are: **Lax**: the browser does not send cookies on cross-site requests, but does send cookies when users navigate to the origin site from an external site. This is the default browser behavior when the **SameSite** value is not specified. **Strict**: the browser sends cookies only for same-site requests. **None**: the browser sends cookies for both cross-site and same-site requests. This value is applicable to re-encrypt and edge routes only. For more information, see the SameSite cookies documentation.	
haproxy.router.openshift.io/set-forwarded-headers	Sets the policy for handling the **Forwarded** and **X-Forwarded-For** HTTP headers per route. The values are: **append**: appends the header, preserving any existing header. This is the default value. **replace**: sets the header, removing any existing header. **never**: never sets the header, but preserves any existing header. **if-none**: sets the header if it is not already set.	**ROUTER_SET_FORWARDED_HEADERS**

1. If the number of IP addresses and CIDR ranges in an allowlist exceeds 61, they are written into a separate file that is then referenced from **haproxy.config**. This file is stored in the **var/lib/haproxy/router/whitelists** folder.

NOTE

To ensure that the addresses are written to the allowlist, check that the full list of CIDR ranges are listed in the Ingress Controller configuration file. The etcd object size limit restricts how large a route annotation can be. Because of this, it creates a threshold for the maximum number of IP addresses and CIDR ranges that you can include in an allowlist.

NOTE

Environment variables cannot be edited.

Router timeout variables

TimeUnits are represented by a number followed by the unit: **us** *(microseconds), **ms** (milliseconds, default), **s** (seconds), **m** (minutes), **h** *(hours), **d** (days).

The regular expression is: [1-9][0-9]*(**us\|ms\|s\|m\|h\|d**).

Variable	Default	Description
ROUTER_BACKEND_CHECK_INTERVAL	5000ms	Length of time between subsequent liveness checks on back ends.
ROUTER_CLIENT_FIN_TIMEOUT	1s	Controls the TCP FIN timeout period for the client connecting to the route. If the FIN sent to close the connection does not answer within the given time, HAProxy closes the connection. This is harmless if set to a low value and uses fewer resources on the router.
ROUTER_DEFAULT_CLIENT_TIMEOUT	30s	Length of time that a client has to acknowledge or send data.
ROUTER_DEFAULT_CONNECT_TIMEOUT	5s	The maximum connection time.
ROUTER_DEFAULT_SERVER_FIN_TIMEOUT	1s	Controls the TCP FIN timeout from the router to the pod backing the route.
ROUTER_DEFAULT_SERVER_TIMEOUT	30s	Length of time that a server has to acknowledge or send data.
ROUTER_DEFAULT_TUNNEL_TIMEOUT	1h	Length of time for TCP or WebSocket connections to remain open. This timeout period resets whenever HAProxy reloads.

Variable	Default	Description
ROUTER_SLOWLORIS_HTTP_KEEPALIVE	**300s**	Set the maximum time to wait for a new HTTP request to appear. If this is set too low, it can cause problems with browsers and applications not expecting a small **keepalive** value.
		Some effective timeout values can be the sum of certain variables, rather than the specific expected timeout. For example, **ROUTER_SLOWLORIS_HTTP_KEEPALIVE** adjusts **timeout http-keep-alive**. It is set to **300s** by default, but HAProxy also waits on **tcp-request inspect-delay**, which is set to **5s**. In this case, the overall timeout would be **300s** plus **5s**.
ROUTER_SLOWLORIS_TIMEOUT	**10s**	Length of time the transmission of an HTTP request can take.
RELOAD_INTERVAL	**5s**	Allows the minimum frequency for the router to reload and accept new changes.
ROUTER_METRICS_HAPROXY_TIMEOUT	**5s**	Timeout for the gathering of HAProxy metrics.

A route setting custom timeout

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  annotations:
    haproxy.router.openshift.io/timeout: 5500ms ❶
...
```

❶ Specifies the new timeout with HAProxy supported units (**us**, **ms**, **s**, **m**, **h**, **d**). If the unit is not provided, **ms** is the default.

> **NOTE**
>
> Setting a server-side timeout value for passthrough routes too low can cause WebSocket connections to timeout frequently on that route.

A route that allows only one specific IP address

```
metadata:
  annotations:
    haproxy.router.openshift.io/ip_whitelist: 192.168.1.10
```

A route that allows several IP addresses

```
metadata:
  annotations:
    haproxy.router.openshift.io/ip_whitelist: 192.168.1.10 192.168.1.11 192.168.1.12
```

A route that allows an IP address CIDR network

```
metadata:
  annotations:
    haproxy.router.openshift.io/ip_whitelist: 192.168.1.0/24
```

A route that allows both IP an address and IP address CIDR networks

```
metadata:
  annotations:
    haproxy.router.openshift.io/ip_whitelist: 180.5.61.153 192.168.1.0/24 10.0.0.0/8
```

A route specifying a rewrite target

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  annotations:
    haproxy.router.openshift.io/rewrite-target: /    1
...
```

1   Sets / as rewrite path of the request on the backend.

Setting the **haproxy.router.openshift.io/rewrite-target** annotation on a route specifies that the Ingress Controller should rewrite paths in HTTP requests using this route before forwarding the requests to the backend application. The part of the request path that matches the path specified in **spec.path** is replaced with the rewrite target specified in the annotation.

The following table provides examples of the path rewriting behavior for various combinations of **spec.path**, request path, and rewrite target.

Table 29.3. rewrite-target examples:

Route.spec.path	Request path	Rewrite target	Forwarded request path
/foo	/foo	/	/
/foo	/foo/	/	/

Route.spec.path	Request path	Rewrite target	Forwarded request path
/foo	/foo/bar	/	/bar
/foo	/foo/bar/	/	/bar/
/foo	/foo	/bar	/bar
/foo	/foo/	/bar	/bar/
/foo	/foo/bar	/baz	/baz/bar
/foo	/foo/bar/	/baz	/baz/bar/
/foo/	/foo	/	N/A (request path does not match route path)
/foo/	/foo/	/	/
/foo/	/foo/bar	/	/bar

29.1.9. Configuring the route admission policy

Administrators and application developers can run applications in multiple namespaces with the same domain name. This is for organizations where multiple teams develop microservices that are exposed on the same hostname.

> **WARNING**
>
> Allowing claims across namespaces should only be enabled for clusters with trust between namespaces, otherwise a malicious user could take over a hostname. For this reason, the default admission policy disallows hostname claims across namespaces.

Prerequisites

- Cluster administrator privileges.

Procedure

- Edit the **.spec.routeAdmission** field of the **ingresscontroller** resource variable using the following command:

```
$ oc -n openshift-ingress-operator patch ingresscontroller/default --patch '{"spec":
{"routeAdmission":{"namespaceOwnership":"InterNamespaceAllowed"}}}' --type=merge
```

–

Sample Ingress Controller configuration

```
spec:
  routeAdmission:
    namespaceOwnership: InterNamespaceAllowed
...
```

TIP

You can alternatively apply the following YAML to configure the route admission policy:

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  routeAdmission:
    namespaceOwnership: InterNamespaceAllowed
```

29.1.10. Creating a route through an Ingress object

Some ecosystem components have an integration with Ingress resources but not with route resources. To cover this case, OpenShift Container Platform automatically creates managed route objects when an Ingress object is created. These route objects are deleted when the corresponding Ingress objects are deleted.

Procedure

1. Define an Ingress object in the OpenShift Container Platform console or by entering the **oc create** command:

   ## YAML Definition of an Ingress

   ```
   apiVersion: networking.k8s.io/v1
   kind: Ingress
   metadata:
     name: frontend
     annotations:
       route.openshift.io/termination: "reencrypt" 1
       route.openshift.io/destination-ca-certificate-secret: secret-ca-cert 2
   spec:
     rules:
     - host: www.example.com 3
       http:
         paths:
         - backend:
             service:
               name: frontend
               port:
                 number: 443
           path: /
   ```

```
      pathType: Prefix
  tls:
  - hosts:
    - www.example.com
    secretName: example-com-tls-certificate
```

1 The **route.openshift.io/termination** annotation can be used to configure the **spec.tls.termination** field of the **Route** as **Ingress** has no field for this. The accepted values are **edge**, **passthrough** and **reencrypt**. All other values are silently ignored. When the annotation value is unset, **edge** is the default route. The TLS certificate details must be defined in the template file to implement the default edge route.

3 When working with an **Ingress** object, you must specify an explicit hostname, unlike when working with routes. You can use the **<host_name>.<cluster_ingress_domain>** syntax, for example **apps.openshiftdemos.com**, to take advantage of the ***.<cluster_ingress_domain>** wildcard DNS record and serving certificate for the cluster. Otherwise, you must ensure that there is a DNS record for the chosen hostname.

    a. If you specify the **passthrough** value in the **route.openshift.io/termination** annotation, set **path** to **''** and **pathType** to **ImplementationSpecific** in the spec:

```
spec:
  rules:
  - host: www.example.com
    http:
      paths:
      - path: ''
        pathType: ImplementationSpecific
        backend:
          service:
            name: frontend
            port:
              number: 443
```

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

2 The **route.openshift.io/destination-ca-certificate-secret** can be used on an Ingress object to define a route with a custom destination certificate (CA). The annotation references a kubernetes secret, **secret-ca-cert** that will be inserted into the generated route.

    a. To specify a route object with a destination CA from an ingress object, you must create a **kubernetes.io/tls** or **Opaque** type secret with a certificate in PEM-encoded format in the **data.tls.crt** specifier of the secret.

2. List your routes:

```
$ oc get routes
```

The result includes an autogenerated route whose name starts with **frontend-**:

```
NAME            HOST/PORT       PATH   SERVICES   PORT   TERMINATION
WILDCARD
frontend-gnztq  www.example.com        frontend   443    reencrypt/Redirect  None
```

–

If you inspect this route, it looks this:

YAML Definition of an autogenerated route

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: frontend-gnztq
  ownerReferences:
  - apiVersion: networking.k8s.io/v1
    controller: true
    kind: Ingress
    name: frontend
    uid: 4e6c59cc-704d-4f44-b390-617d879033b6
spec:
  host: www.example.com
  path: /
  port:
    targetPort: https
  tls:
    certificate: |
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
    insecureEdgeTerminationPolicy: Redirect
    key: |
      -----BEGIN RSA PRIVATE KEY-----
      [...]
      -----END RSA PRIVATE KEY-----
    termination: reencrypt
    destinationCACertificate: |
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
  to:
    kind: Service
    name: frontend
```

29.1.11. Creating a route using the default certificate through an Ingress object

If you create an Ingress object without specifying any TLS configuration, OpenShift Container Platform generates an insecure route. To create an Ingress object that generates a secure, edge-terminated route using the default ingress certificate, you can specify an empty TLS configuration as follows.

Prerequisites

- You have a service that you want to expose.

- You have access to the OpenShift CLI (**oc**).

Procedure

1. Create a YAML file for the Ingress object. In this example, the file is called **example-ingress.yaml**:

YAML definition of an Ingress object

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: frontend
  ...
spec:
  rules:
    ...
  tls:
  - {} 1
```

1 Use this exact syntax to specify TLS without specifying a custom certificate.

2. Create the Ingress object by running the following command:

```
$ oc create -f example-ingress.yaml
```

Verification

- Verify that OpenShift Container Platform has created the expected route for the Ingress object by running the following command:

```
$ oc get routes -o yaml
```

Example output

```
apiVersion: v1
items:
- apiVersion: route.openshift.io/v1
  kind: Route
  metadata:
    name: frontend-j9sdd 1
    ...
  spec:
  ...
    tls: 2
      insecureEdgeTerminationPolicy: Redirect
      termination: edge 3
  ...
```

1 The name of the route includes the name of the Ingress object followed by a random suffix.

2 In order to use the default certificate, the route should not specify **spec.certificate**.

3 The route should specify the **edge** termination policy.

29.1.12. Creating a route using the destination CA certificate in the Ingress annotation

The **route.openshift.io/destination-ca-certificate-secret** annotation can be used on an Ingress object to define a route with a custom destination CA certificate.

Prerequisites

- You may have a certificate/key pair in PEM-encoded files, where the certificate is valid for the route host.

- You may have a separate CA certificate in a PEM-encoded file that completes the certificate chain.

- You must have a separate destination CA certificate in a PEM-encoded file.

- You must have a service that you want to expose.

Procedure

1. Add the **route.openshift.io/destination-ca-certificate-secret** to the Ingress annotations:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: frontend
  annotations:
    route.openshift.io/termination: "reencrypt"
    route.openshift.io/destination-ca-certificate-secret: secret-ca-cert ❶
...
```

❶ The annotation references a kubernetes secret.

2. The secret referenced in this annotation will be inserted into the generated route.

   Example output

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: frontend
  annotations:
    route.openshift.io/termination: reencrypt
    route.openshift.io/destination-ca-certificate-secret: secret-ca-cert
spec:
...
  tls:
    insecureEdgeTerminationPolicy: Redirect
    termination: reencrypt
    destinationCACertificate: |
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
...
```

29.1.13. Configuring the OpenShift Container Platform Ingress Controller for dual-stack networking

If your OpenShift Container Platform cluster is configured for IPv4 and IPv6 dual-stack networking, your cluster is externally reachable by OpenShift Container Platform routes.

The Ingress Controller automatically serves services that have both IPv4 and IPv6 endpoints, but you can configure the Ingress Controller for single-stack or dual-stack services.

Prerequisites

- You deployed an OpenShift Container Platform cluster on bare metal.

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

Procedure

1. To have the Ingress Controller serve traffic over IPv4/IPv6 to a workload, you can create a service YAML file or modify an existing service YAML file by setting the **ipFamilies** and **ipFamilyPolicy** fields. For example:

   Sample service YAML file

   ```
   apiVersion: v1
   kind: Service
   metadata:
     creationTimestamp: yyyy-mm-ddT00:00:00Z
     labels:
       name: <service_name>
       manager: kubectl-create
       operation: Update
       time: yyyy-mm-ddT00:00:00Z
     name: <service_name>
     namespace: <namespace_name>
     resourceVersion: "<resource_version_number>"
     selfLink: "/api/v1/namespaces/<namespace_name>/services/<service_name>"
     uid: <uid_number>
   spec:
     clusterIP: 172.30.0.0/16
     clusterIPs:          1
     - 172.30.0.0/16
     - <second_IP_address>
     ipFamilies:          2
     - IPv4
     - IPv6
     ipFamilyPolicy: RequireDualStack          3
     ports:
     - port: 8080
       protocol: TCP
       targetport: 8080
     selector:
       name: <namespace_name>
     sessionAffinity: None
   ```

```
    type: ClusterIP
status:
  loadbalancer: {}
```

1 In a dual-stack instance, there are two different **clusterIPs** provided.

2 For a single-stack instance, enter **IPv4** or **IPv6**. For a dual-stack instance, enter both **IPv4** and **IPv6**.

3 For a single-stack instance, enter **SingleStack**. For a dual-stack instance, enter **RequireDualStack**.

These resources generate corresponding **endpoints**. The Ingress Controller now watches **endpointslices**.

2. To view **endpoints**, enter the following command:

```
$ oc get endpoints
```

3. To view **endpointslices**, enter the following command:

```
$ oc get endpointslices
```

Additional resources

- Specifying an alternative cluster domain using the appsDomain option

29.2. SECURED ROUTES

Secure routes provide the ability to use several types of TLS termination to serve certificates to the client. The following sections describe how to create re-encrypt, edge, and passthrough routes with custom certificates.

> **IMPORTANT**
>
> If you create routes in Microsoft Azure through public endpoints, the resource names are subject to restriction. You cannot create resources that use certain terms. For a list of terms that Azure restricts, see Resolve reserved resource name errors in the Azure documentation.

29.2.1. Creating a re-encrypt route with a custom certificate

You can configure a secure route using reencrypt TLS termination with a custom certificate by using the **oc create route** command.

Prerequisites

- You must have a certificate/key pair in PEM-encoded files, where the certificate is valid for the route host.

- You may have a separate CA certificate in a PEM-encoded file that completes the certificate chain.

- You must have a separate destination CA certificate in a PEM-encoded file.

- You must have a service that you want to expose.

NOTE

Password protected key files are not supported. To remove a passphrase from a key file, use the following command:

```
$ openssl rsa -in password_protected_tls.key -out tls.key
```

Procedure

This procedure creates a **Route** resource with a custom certificate and reencrypt TLS termination. The following assumes that the certificate/key pair are in the **tls.crt** and **tls.key** files in the current working directory. You must also specify a destination CA certificate to enable the Ingress Controller to trust the service's certificate. You may also specify a CA certificate if needed to complete the certificate chain. Substitute the actual path names for **tls.crt**, **tls.key**, **cacert.crt**, and (optionally) **ca.crt**. Substitute the name of the **Service** resource that you want to expose for **frontend**. Substitute the appropriate hostname for **www.example.com**.

- Create a secure **Route** resource using reencrypt TLS termination and a custom certificate:

```
$ oc create route reencrypt --service=frontend --cert=tls.crt --key=tls.key --dest-ca-cert=destca.crt --ca-cert=ca.crt --hostname=www.example.com
```

If you examine the resulting **Route** resource, it should look similar to the following:

YAML Definition of the Secure Route

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: frontend
spec:
  host: www.example.com
  to:
    kind: Service
    name: frontend
  tls:
    termination: reencrypt
    key: |-
      -----BEGIN PRIVATE KEY-----
      [...]
      -----END PRIVATE KEY-----
    certificate: |-
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
    caCertificate: |-
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
    destinationCACertificate: |-
```

```
-----BEGIN CERTIFICATE-----
[...]
-----END CERTIFICATE-----
```

See **oc create route reencrypt --help** for more options.

29.2.2. Creating an edge route with a custom certificate

You can configure a secure route using edge TLS termination with a custom certificate by using the **oc create route** command. With an edge route, the Ingress Controller terminates TLS encryption before forwarding traffic to the destination pod. The route specifies the TLS certificate and key that the Ingress Controller uses for the route.

Prerequisites

- You must have a certificate/key pair in PEM-encoded files, where the certificate is valid for the route host.

- You may have a separate CA certificate in a PEM-encoded file that completes the certificate chain.

- You must have a service that you want to expose.

> **NOTE**
>
> Password protected key files are not supported. To remove a passphrase from a key file, use the following command:
>
> ```
> $ openssl rsa -in password_protected_tls.key -out tls.key
> ```

Procedure

This procedure creates a **Route** resource with a custom certificate and edge TLS termination. The following assumes that the certificate/key pair are in the **tls.crt** and **tls.key** files in the current working directory. You may also specify a CA certificate if needed to complete the certificate chain. Substitute the actual path names for **tls.crt**, **tls.key**, and (optionally) **ca.crt**. Substitute the name of the service that you want to expose for **frontend**. Substitute the appropriate hostname for **www.example.com**.

- Create a secure **Route** resource using edge TLS termination and a custom certificate.

  ```
  $ oc create route edge --service=frontend --cert=tls.crt --key=tls.key --ca-cert=ca.crt --hostname=www.example.com
  ```

  If you examine the resulting **Route** resource, it should look similar to the following:

  YAML Definition of the Secure Route

  ```
  apiVersion: route.openshift.io/v1
  kind: Route
  metadata:
    name: frontend
  spec:
    host: www.example.com
    to:
  ```

```
  kind: Service
  name: frontend
tls:
  termination: edge
  key: |-
    -----BEGIN PRIVATE KEY-----
    [...]
    -----END PRIVATE KEY-----
  certificate: |-
    -----BEGIN CERTIFICATE-----
    [...]
    -----END CERTIFICATE-----
  caCertificate: |-
    -----BEGIN CERTIFICATE-----
    [...]
    -----END CERTIFICATE-----
```

See **oc create route edge --help** for more options.

29.2.3. Creating a passthrough route

You can configure a secure route using passthrough termination by using the **oc create route** command. With passthrough termination, encrypted traffic is sent straight to the destination without the router providing TLS termination. Therefore no key or certificate is required on the route.

Prerequisites

- You must have a service that you want to expose.

Procedure

- Create a **Route** resource:

  ```
  $ oc create route passthrough route-passthrough-secured --service=frontend --port=8080
  ```

  If you examine the resulting **Route** resource, it should look similar to the following:

  #### A Secured Route Using Passthrough Termination

  ```
  apiVersion: route.openshift.io/v1
  kind: Route
  metadata:
    name: route-passthrough-secured ❶
  spec:
    host: www.example.com
    port:
      targetPort: 8080
    tls:
      termination: passthrough ❷
      insecureEdgeTerminationPolicy: None ❸
    to:
      kind: Service
      name: frontend
  ```

1. The name of the object, which is limited to 63 characters.

2. The **termination** field is set to **passthrough**. This is the only required **tls** field.

3. Optional **insecureEdgeTerminationPolicy**. The only valid values are **None**, **Redirect**, or empty for disabled.

The destination pod is responsible for serving certificates for the traffic at the endpoint. This is currently the only method that can support requiring client certificates, also known as two-way authentication.

CHAPTER 30. CONFIGURING INGRESS CLUSTER TRAFFIC

30.1. CONFIGURING INGRESS CLUSTER TRAFFIC OVERVIEW

OpenShift Container Platform provides the following methods for communicating from outside the cluster with services running in the cluster.

The methods are recommended, in order or preference:

- If you have HTTP/HTTPS, use an Ingress Controller.

- If you have a TLS-encrypted protocol other than HTTPS. For example, for TLS with the SNI header, use an Ingress Controller.

- Otherwise, use a Load Balancer, an External IP, or a **NodePort**.

Method	Purpose
Use an Ingress Controller	Allows access to HTTP/HTTPS traffic and TLS-encrypted protocols other than HTTPS (for example, TLS with the SNI header).
Automatically assign an external IP using a load balancer service	Allows traffic to non-standard ports through an IP address assigned from a pool. Most cloud platforms offer a method to start a service with a load-balancer IP address.
About MetalLB and the MetalLB Operator	Allows traffic to a specific IP address or address from a pool on the machine network. For bare-metal installations or platforms that are like bare metal, MetalLB provides a way to start a service with a load-balancer IP address.
Manually assign an external IP to a service	Allows traffic to non-standard ports through a specific IP address.
Configure a **NodePort**	Expose a service on all nodes in the cluster.

30.1.1. Comparision: Fault tolerant access to external IP addresses

For the communication methods that provide access to an external IP address, fault tolerant access to the IP address is another consideration. The following features provide fault tolerant access to an external IP address.

IP failover

IP failover manages a pool of virtual IP address for a set of nodes. It is implemented with Keepalived and Virtual Router Redundancy Protocol (VRRP). IP failover is a layer 2 mechanism only and relies on multicast. Multicast can have disadvantages for some networks.

MetalLB

MetalLB has a layer 2 mode, but it does not use multicast. Layer 2 mode has a disadvantage that it transfers all traffic for an external IP address through one node.

Manually assigning external IP addresses

You can configure your cluster with an IP address block that is used to assign external IP addresses to services. By default, this feature is disabled. This feature is flexible, but places the largest burden on the cluster or network administrator. The cluster is prepared to receive traffic that is destined for the external IP, but each customer has to decide how they want to route traffic to nodes.

30.2. CONFIGURING EXTERNALIPS FOR SERVICES

As a cluster administrator, you can designate an IP address block that is external to the cluster that can send traffic to services in the cluster.

This functionality is generally most useful for clusters installed on bare-metal hardware.

30.2.1. Prerequisites

- Your network infrastructure must route traffic for the external IP addresses to your cluster.

30.2.2. About ExternalIP

For non-cloud environments, OpenShift Container Platform supports the assignment of external IP addresses to a **Service** object **spec.externalIPs[]** field through the **ExternalIP** facility. By setting this field, OpenShift Container Platform assigns an additional virtual IP address to the service. The IP address can be outside the service network defined for the cluster. A service configured with an ExternalIP functions similarly to a service with **type=NodePort**, allowing you to direct traffic to a local node for load balancing.

You must configure your networking infrastructure to ensure that the external IP address blocks that you define are routed to the cluster. As a result, the IP address is not configured in the network interfaces from nodes. To handle the traffic, you must configure the routing and access to the external IP by using a method such as static Address Resolution Protocol (ARP) entries.

OpenShift Container Platform extends the ExternalIP functionality in Kubernetes by adding the following capabilities:

- Restrictions on the use of external IP addresses by users through a configurable policy

- Allocation of an external IP address automatically to a service upon request

> ⚠️ **WARNING**
>
> Disabled by default, use of ExternalIP functionality can be a security risk, because in-cluster traffic to an external IP address is directed to that service. This could allow cluster users to intercept sensitive traffic destined for external resources.

> **IMPORTANT**
>
> This feature is supported only in non-cloud deployments. For cloud deployments, use the load balancer services for automatic deployment of a cloud load balancer to target the endpoints of a service.

You can assign an external IP address in the following ways:

Automatic assignment of an external IP

OpenShift Container Platform automatically assigns an IP address from the **autoAssignCIDRs** CIDR block to the **spec.externalIPs[]** array when you create a **Service** object with **spec.type=LoadBalancer** set. In this case, OpenShift Container Platform implements a non-cloud version of the load balancer service type and assigns IP addresses to the services. Automatic assignment is disabled by default and must be configured by a cluster administrator as described in the following section.

Manual assignment of an external IP

OpenShift Container Platform uses the IP addresses assigned to the **spec.externalIPs[]** array when you create a **Service** object. You cannot specify an IP address that is already in use by another service.

30.2.2.1. Configuration for ExternalIP

Use of an external IP address in OpenShift Container Platform is governed by the following fields in the **Network.config.openshift.io** CR named **cluster**:

- **spec.externalIP.autoAssignCIDRs** defines an IP address block used by the load balancer when choosing an external IP address for the service. OpenShift Container Platform supports only a single IP address block for automatic assignment. This can be simpler than having to manage the port space of a limited number of shared IP addresses when manually assigning ExternalIPs to services. If automatic assignment is enabled, a **Service** object with **spec.type=LoadBalancer** is allocated an external IP address.

- **spec.externalIP.policy** defines the permissible IP address blocks when manually specifying an IP address. OpenShift Container Platform does not apply policy rules to IP address blocks defined by **spec.externalIP.autoAssignCIDRs**.

If routed correctly, external traffic from the configured external IP address block can reach service endpoints through any TCP or UDP port that the service exposes.

> **IMPORTANT**
>
> As a cluster administrator, you must configure routing to externalIPs on both OpenShiftSDN and OVN-Kubernetes network types. You must also ensure that the IP address block you assign terminates at one or more nodes in your cluster. For more information, see [Kubernetes External IPs](#).

OpenShift Container Platform supports both the automatic and manual assignment of IP addresses, and each address is guaranteed to be assigned to a maximum of one service. This ensures that each service can expose its chosen ports regardless of the ports exposed by other services.

> **NOTE**
>
> To use IP address blocks defined by **autoAssignCIDRs** in OpenShift Container Platform, you must configure the necessary IP address assignment and routing for your host network.

The following YAML describes a service with an external IP address configured:

Example Service object with `spec.externalIPs[]` set

```
apiVersion: v1
kind: Service
metadata:
  name: http-service
spec:
  clusterIP: 172.30.163.110
  externalIPs:
  - 192.168.132.253
  externalTrafficPolicy: Cluster
  ports:
  - name: highport
    nodePort: 31903
    port: 30102
    protocol: TCP
    targetPort: 30102
  selector:
    app: web
  sessionAffinity: None
  type: LoadBalancer
status:
  loadBalancer:
    ingress:
    - ip: 192.168.132.253
```

30.2.2.2. Restrictions on the assignment of an external IP address

As a cluster administrator, you can specify IP address blocks to allow and to reject.

Restrictions apply only to users without **cluster-admin** privileges. A cluster administrator can always set the service **spec.externalIPs[]** field to any IP address.

You configure IP address policy with a **policy** object defined by specifying the **spec.ExternalIP.policy** field. The policy object has the following shape:

```
{
  "policy": {
    "allowedCIDRs": [],
    "rejectedCIDRs": []
  }
}
```

When configuring policy restrictions, the following rules apply:

- If **policy={}** is set, then creating a **Service** object with **spec.ExternalIPs[]** set will fail. This is the default for OpenShift Container Platform. The behavior when **policy=null** is set is identical.

- If **policy** is set and either **policy.allowedCIDRs[]** or **policy.rejectedCIDRs[]** is set, the following rules apply:

  - If **allowedCIDRs[]** and **rejectedCIDRs[]** are both set, then **rejectedCIDRs[]** has precedence over **allowedCIDRs[]**.

  - If **allowedCIDRs[]** is set, creating a **Service** object with **spec.ExternalIPs[]** will succeed only if the specified IP addresses are allowed.

- If **rejectedCIDRs[]** is set, creating a **Service** object with **spec.ExternalIPs[]** will succeed only if the specified IP addresses are not rejected.

30.2.2.3. Example policy objects

The examples that follow demonstrate several different policy configurations.

- In the following example, the policy prevents OpenShift Container Platform from creating any service with an external IP address specified:

  Example policy to reject any value specified for Service object spec.externalIPs[]

  ```
  apiVersion: config.openshift.io/v1
  kind: Network
  metadata:
    name: cluster
  spec:
    externalIP:
      policy: {}
    ...
  ```

- In the following example, both the **allowedCIDRs** and **rejectedCIDRs** fields are set.

  Example policy that includes both allowed and rejected CIDR blocks

  ```
  apiVersion: config.openshift.io/v1
  kind: Network
  metadata:
    name: cluster
  spec:
    externalIP:
      policy:
        allowedCIDRs:
        - 172.16.66.10/23
        rejectedCIDRs:
        - 172.16.66.10/24
    ...
  ```

- In the following example, **policy** is set to **null**. If set to **null**, when inspecting the configuration object by entering **oc get networks.config.openshift.io -o yaml**, the **policy** field will not appear in the output.

  Example policy to allow any value specified for Service object spec.externalIPs[]

  ```
  apiVersion: config.openshift.io/v1
  kind: Network
  metadata:
    name: cluster
  spec:
    externalIP:
      policy: null
    ...
  ```

30.2.3. ExternalIP address block configuration

The configuration for ExternalIP address blocks is defined by a Network custom resource (CR) named
cluster. The Network CR is part of the **config.openshift.io** API group.

> **IMPORTANT**
>
> During cluster installation, the Cluster Version Operator (CVO) automatically creates a
> Network CR named **cluster**. Creating any other CR objects of this type is not supported.

The following YAML describes the ExternalIP configuration:

Network.config.openshift.io CR named **cluster**

```
apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
 externalIP:
   autoAssignCIDRs: []  1
   policy:  2
     ...
```

1 Defines the IP address block in CIDR format that is available for automatic assignment of external
IP addresses to a service. Only a single IP address range is allowed.

2 Defines restrictions on manual assignment of an IP address to a service. If no restrictions are
defined, specifying the **spec.externalIP** field in a **Service** object is not allowed. By default, no
restrictions are defined.

The following YAML describes the fields for the **policy** stanza:

Network.config.openshift.io **policy** stanza

```
policy:
  allowedCIDRs: []  1
  rejectedCIDRs: []  2
```

1 A list of allowed IP address ranges in CIDR format.

2 A list of rejected IP address ranges in CIDR format.

Example external IP configurations
Several possible configurations for external IP address pools are displayed in the following examples:

- The following YAML describes a configuration that enables automatically assigned external IP
addresses:

  Example configuration with spec.externalIP.autoAssignCIDRs set

  ```
  apiVersion: config.openshift.io/v1
  ```

```
kind: Network
metadata:
  name: cluster
spec:
  ...
  externalIP:
    autoAssignCIDRs:
    - 192.168.132.254/29
```

- The following YAML configures policy rules for the allowed and rejected CIDR ranges:

Example configuration with spec.externalIP.policy set

```
apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  ...
  externalIP:
    policy:
      allowedCIDRs:
      - 192.168.132.0/29
      - 192.168.132.8/29
      rejectedCIDRs:
      - 192.168.132.7/32
```

30.2.4. Configure external IP address blocks for your cluster

As a cluster administrator, you can configure the following ExternalIP settings:

- An ExternalIP address block used by OpenShift Container Platform to automatically populate the **spec.clusterIP** field for a **Service** object.

- A policy object to restrict what IP addresses may be manually assigned to the **spec.clusterIP** array of a **Service** object.

Prerequisites

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

- Access to the cluster as a user with the **cluster-admin** role.

Procedure

1. Optional: To display the current external IP configuration, enter the following command:

   ```
   $ oc describe networks.config cluster
   ```

2. To edit the configuration, enter the following command:

   ```
   $ oc edit networks.config cluster
   ```

3. Modify the ExternalIP configuration, as in the following example:

```
apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  ...
  externalIP: 1
  ...
```

1   Specify the configuration for the **externalIP** stanza.

4. To confirm the updated ExternalIP configuration, enter the following command:

```
$ oc get networks.config cluster -o go-template='{{.spec.externalIP}}{{"\n"}}'
```

30.2.5. Next steps

- Configuring ingress cluster traffic for a service external IP

30.3. CONFIGURING INGRESS CLUSTER TRAFFIC USING AN INGRESS CONTROLLER

OpenShift Container Platform provides methods for communicating from outside the cluster with services running in the cluster. This method uses an Ingress Controller.

30.3.1. Using Ingress Controllers and routes

The Ingress Operator manages Ingress Controllers and wildcard DNS.

Using an Ingress Controller is the most common way to allow external access to an OpenShift Container Platform cluster.

An Ingress Controller is configured to accept external requests and proxy them based on the configured routes. This is limited to HTTP, HTTPS using SNI, and TLS using SNI, which is sufficient for web applications and services that work over TLS with SNI.

Work with your administrator to configure an Ingress Controller to accept external requests and proxy them based on the configured routes.

The administrator can create a wildcard DNS entry and then set up an Ingress Controller. Then, you can work with the edge Ingress Controller without having to contact the administrators.

By default, every Ingress Controller in the cluster can admit any route created in any project in the cluster.

The Ingress Controller:

- Has two replicas by default, which means it should be running on two worker nodes.

- Can be scaled up to have more replicas on more nodes.

NOTE

The procedures in this section require prerequisites performed by the cluster administrator.

30.3.2. Prerequisites

Before starting the following procedures, the administrator must:

- Set up the external port to the cluster networking environment so that requests can reach the cluster.

- Make sure there is at least one user with cluster admin role. To add this role to a user, run the following command:

```
$ oc adm policy add-cluster-role-to-user cluster-admin username
```

- Have an OpenShift Container Platform cluster with at least one master and at least one node and a system outside the cluster that has network access to the cluster. This procedure assumes that the external system is on the same subnet as the cluster. The additional networking required for external systems on a different subnet is out–of–scope for this topic.

30.3.3. Creating a project and service

If the project and service that you want to expose do not exist, first create the project, then the service.

If the project and service already exist, skip to the procedure on exposing the service to create a route.

Prerequisites

- Install the **oc** CLI and log in as a cluster administrator.

Procedure

1. Create a new project for your service by running the **oc new-project** command:

```
$ oc new-project myproject
```

2. Use the **oc new-app** command to create your service:

```
$ oc new-app nodejs:12~https://github.com/sclorg/nodejs-ex.git
```

3. To verify that the service was created, run the following command:

```
$ oc get svc -n myproject
```

Example output

```
NAME       TYPE       CLUSTER-IP      EXTERNAL-IP  PORT(S)   AGE
nodejs-ex  ClusterIP  172.30.197.157  <none>       8080/TCP  70s
```

By default, the new service does not have an external IP address.

30.3.4. Exposing the service by creating a route

You can expose the service as a route by using the **oc expose** command.

Procedure

To expose the service:

1. Log in to OpenShift Container Platform.

2. Log in to the project where the service you want to expose is located:

   ```
   $ oc project myproject
   ```

3. Run the **oc expose service** command to expose the route:

   ```
   $ oc expose service nodejs-ex
   ```

   #### Example output

   ```
   route.route.openshift.io/nodejs-ex exposed
   ```

4. To verify that the service is exposed, you can use a tool, such as cURL, to make sure the service is accessible from outside the cluster.

   a. Use the **oc get route** command to find the route's host name:

      ```
      $ oc get route
      ```

      #### Example output

      ```
      NAME       HOST/PORT                    PATH   SERVICES   PORT      TERMINATION   WILDCARD
      nodejs-ex   nodejs-ex-myproject.example.com        nodejs-ex   8080-tcp                 None
      ```

   b. Use cURL to check that the host responds to a GET request:

      ```
      $ curl --head nodejs-ex-myproject.example.com
      ```

      #### Example output

      ```
      HTTP/1.1 200 OK
      ...
      ```

30.3.5. Configuring Ingress Controller sharding by using route labels

Ingress Controller sharding by using route labels means that the Ingress Controller serves any route in any namespace that is selected by the route selector.

Figure 30.1. Ingress sharding using route labels

Ingress Controller sharding is useful when balancing incoming traffic load among a set of Ingress Controllers and when isolating traffic to a specific Ingress Controller. For example, company A goes to one Ingress Controller and company B to another.

Procedure

1. Edit the **router-internal.yaml** file:

```
# cat router-internal.yaml
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: sharded
  namespace: openshift-ingress-operator
spec:
  domain: <apps-sharded.basedomain.example.net> 1
  nodePlacement:
    nodeSelector:
      matchLabels:
        node-role.kubernetes.io/worker: ""
  routeSelector:
    matchLabels:
      type: sharded
```

1 Specify a domain to be used by the Ingress Controller. This domain must be different from the default Ingress Controller domain.

2. Apply the Ingress Controller **router-internal.yaml** file:

```
# oc apply -f router-internal.yaml
```

The Ingress Controller selects routes in any namespace that have the label **type: sharded**.

3. Create a new route using the domain configured in the **router-internal.yaml**:

```
$ oc expose svc <service-name> --hostname <route-name>.apps-
sharded.basedomain.example.net
```

30.3.6. Configuring Ingress Controller sharding by using namespace labels

Ingress Controller sharding by using namespace labels means that the Ingress Controller serves any route in any namespace that is selected by the namespace selector.

Figure 30.2. Ingress sharding using namespace labels

Ingress Controller sharding is useful when balancing incoming traffic load among a set of Ingress Controllers and when isolating traffic to a specific Ingress Controller. For example, company A goes to one Ingress Controller and company B to another.

Procedure

1. Edit the **router-internal.yaml** file:

```
# cat router-internal.yaml
```

Example output

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: sharded
  namespace: openshift-ingress-operator
spec:
  domain: <apps-sharded.basedomain.example.net> 1
  nodePlacement:
    nodeSelector:
      matchLabels:
        node-role.kubernetes.io/worker: ""
  namespaceSelector:
    matchLabels:
      type: sharded
```

1   Specify a domain to be used by the Ingress Controller. This domain must be different from the default Ingress Controller domain.

2. Apply the Ingress Controller **router-internal.yaml** file:

   ```
   # oc apply -f router-internal.yaml
   ```

   The Ingress Controller selects routes in any namespace that is selected by the namespace selector that have the label **type: sharded**.

3. Create a new route using the domain configured in the **router-internal.yaml**:

   ```
   $ oc expose svc <service-name> --hostname <route-name>.apps-
   sharded.basedomain.example.net
   ```

30.3.7. Creating a route for Ingress Controller sharding

A route allows you to host your application at a URL. In this case, the hostname is not set and the route uses a subdomain instead. When you specify a subdomain, you automatically use the domain of the Ingress Controller that exposes the route. For situations where a route is exposed by multiple Ingress Controllers, the route is hosted at multiple URLs.

The following procedure describes how to create a route for Ingress Controller sharding, using the **hello-openshift** application as an example.

Ingress Controller sharding is useful when balancing incoming traffic load among a set of Ingress Controllers and when isolating traffic to a specific Ingress Controller. For example, company A goes to one Ingress Controller and company B to another.

Prerequisites

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

- You are logged in as a project administrator.

- You have a web application that exposes a port and an HTTP or TLS endpoint listening for traffic on the port.

- You have configured the Ingress Controller for sharding.

Procedure

1. Create a project called **hello-openshift** by running the following command:

   ```
   $ oc new-project hello-openshift
   ```

2. Create a pod in the project by running the following command:

   ```
   $ oc create -f https://raw.githubusercontent.com/openshift/origin/master/examples/hello-openshift/hello-pod.json
   ```

3. Create a service called **hello-openshift** by running the following command:

   ```
   $ oc expose pod/hello-openshift
   ```

4. Create a route definition called **hello-openshift-route.yaml**:

   YAML definition of the created route for sharding:

   ```
   apiVersion: route.openshift.io/v1
   kind: Route
   metadata:
     labels:
       type: sharded       1
     name: hello-openshift-edge
     namespace: hello-openshift
   spec:
     subdomain: hello-openshift   2
     tls:
       termination: edge
     to:
       kind: Service
       name: hello-openshift
   ```

   1   Both the label key and its corresponding label value must match the ones specified in the Ingress Controller. In this example, the Ingress Controller has the label key and value **type: sharded**.

   2   The route will be exposed using the value of the **subdomain** field. When you specify the **subdomain** field, you must leave the hostname unset. If you specify both the **host** and **subdomain** fields, then the route will use the value of the **host** field, and ignore the **subdomain** field.

5. Use **hello-openshift-route.yaml** to create a route to the **hello-openshift** application by running the following command:

   ```
   $ oc -n hello-openshift create -f hello-openshift-route.yaml
   ```

Verification

- Get the status of the route with the following command:

   ```
   $ oc -n hello-openshift get routes/hello-openshift-edge -o yaml
   ```

The resulting **Route** resource should look similar to the following:

Example output

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  labels:
    type: sharded
  name: hello-openshift-edge
  namespace: hello-openshift
spec:
  subdomain: hello-openshift
  tls:
    termination: edge
  to:
    kind: Service
    name: hello-openshift
status:
  ingress:
  - host: hello-openshift.<apps-sharded.basedomain.example.net>  1
    routerCanonicalHostname: router-sharded.<apps-sharded.basedomain.example.net>  2
    routerName: sharded  3
```

1 The hostname the Ingress Controller, or router, uses to expose the route. The value of the **host** field is automatically determined by the Ingress Controller, and uses its domain. In this example, the domain of the Ingress Controller is **<apps-sharded.basedomain.example.net>**.

2 The hostname of the Ingress Controller.

3 The name of the Ingress Controller. In this example, the Ingress Controller has the name **sharded**.

30.3.8. Additional resources

The Ingress Operator manages wildcard DNS. For more information, see the following:

- Ingress Operator in OpenShift Container Platform .

- Installing a cluster on bare metal .

- Installing a cluster on vSphere .

- About network policy .

30.4. CONFIGURING INGRESS CLUSTER TRAFFIC USING A LOAD BALANCER

OpenShift Container Platform provides methods for communicating from outside the cluster with services running in the cluster. This method uses a load balancer.

30.4.1. Using a load balancer to get traffic into the cluster

If you do not need a specific external IP address, you can configure a load balancer service to allow external access to an OpenShift Container Platform cluster.

A load balancer service allocates a unique IP. The load balancer has a single edge router IP, which can be a virtual IP (VIP), but is still a single machine for initial load balancing.
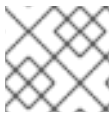
> **NOTE**
>
> If a pool is configured, it is done at the infrastructure level, not by a cluster administrator.

> **NOTE**
>
> The procedures in this section require prerequisites performed by the cluster administrator.

30.4.2. Prerequisites

Before starting the following procedures, the administrator must:

- Set up the external port to the cluster networking environment so that requests can reach the cluster.

- Make sure there is at least one user with cluster admin role. To add this role to a user, run the following command:

  ```
  $ oc adm policy add-cluster-role-to-user cluster-admin username
  ```

- Have an OpenShift Container Platform cluster with at least one master and at least one node and a system outside the cluster that has network access to the cluster. This procedure assumes that the external system is on the same subnet as the cluster. The additional networking required for external systems on a different subnet is out-of-scope for this topic.

30.4.3. Creating a project and service

If the project and service that you want to expose do not exist, first create the project, then the service.

If the project and service already exist, skip to the procedure on exposing the service to create a route.

Prerequisites

- Install the **oc** CLI and log in as a cluster administrator.

Procedure

1. Create a new project for your service by running the **oc new-project** command:

   ```
   $ oc new-project myproject
   ```

2. Use the **oc new-app** command to create your service:

   ```
   $ oc new-app nodejs:12~https://github.com/sclorg/nodejs-ex.git
   ```

3. To verify that the service was created, run the following command:

```
$ oc get svc -n myproject
```

Example output

```
NAME       TYPE       CLUSTER-IP      EXTERNAL-IP  PORT(S)   AGE
nodejs-ex  ClusterIP  172.30.197.157  <none>       8080/TCP  70s
```

By default, the new service does not have an external IP address.

30.4.4. Exposing the service by creating a route

You can expose the service as a route by using the **oc expose** command.

Procedure

To expose the service:

1. Log in to OpenShift Container Platform.

2. Log in to the project where the service you want to expose is located:

```
$ oc project myproject
```

3. Run the **oc expose service** command to expose the route:

```
$ oc expose service nodejs-ex
```

Example output

```
route.route.openshift.io/nodejs-ex exposed
```

4. To verify that the service is exposed, you can use a tool, such as cURL, to make sure the service is accessible from outside the cluster.

   a. Use the **oc get route** command to find the route's host name:

```
$ oc get route
```

   Example output

```
NAME       HOST/PORT                      PATH  SERVICES   PORT      TERMINATION
WILDCARD
nodejs-ex  nodejs-ex-myproject.example.com      nodejs-ex  8080-tcp              None
```

   b. Use cURL to check that the host responds to a GET request:

```
$ curl --head nodejs-ex-myproject.example.com
```

   Example output

```
HTTP/1.1 200 OK
...
```

30.4.5. Creating a load balancer service

Use the following procedure to create a load balancer service.

Prerequisites

- Make sure that the project and service you want to expose exist.

- Your cloud provider supports load balancers.

Procedure

To create a load balancer service:

1. Log in to OpenShift Container Platform.

2. Load the project where the service you want to expose is located.

   ```
   $ oc project project1
   ```

3. Open a text file on the control plane node and paste the following text, editing the file as needed:

   Sample load balancer configuration file

   ```
   apiVersion: v1
   kind: Service
   metadata:
     name: egress-2 1
   spec:
     ports:
     - name: db
       port: 3306 2
     loadBalancerIP:
     loadBalancerSourceRanges: 3
     - 10.0.0.0/8
     - 192.168.0.0/16
     type: LoadBalancer 4
     selector:
       name: mysql 5
   ```

   1   Enter a descriptive name for the load balancer service.

   2   Enter the same port that the service you want to expose is listening on.

   3   Enter a list of specific IP addresses to restrict traffic through the load balancer. This field is ignored if the cloud-provider does not support the feature.

   4   Enter **Loadbalancer** as the type.

   5   Enter the name of the service.

> **NOTE**
>
> To restrict the traffic through the load balancer to specific IP addresses, it is recommended to use the Ingress Controller field **spec.endpointPublishingStrategy.loadBalancer.allowedSourceRanges**. Do not set the **loadBalancerSourceRanges** field.

4. Save and exit the file.

5. Run the following command to create the service:

   ```
   $ oc create -f <file-name>
   ```

   For example:

   ```
   $ oc create -f mysql-lb.yaml
   ```

6. Execute the following command to view the new service:

   ```
   $ oc get svc
   ```

   Example output

   ```
   NAME      TYPE          CLUSTER-IP     EXTERNAL-IP                          PORT(S)
   AGE
   egress-2  LoadBalancer  172.30.22.226  ad42f5d8b303045-487804948.example.com
   3306:30357/TCP   15m
   ```

   The service has an external IP address automatically assigned if there is a cloud provider enabled.

7. On the master, use a tool, such as cURL, to make sure you can reach the service using the public IP address:

   ```
   $ curl <public-ip>:<port>
   ```

   For example:

   ```
   $ curl 172.29.121.74:3306
   ```

   The examples in this section use a MySQL service, which requires a client application. If you get a string of characters with the **Got packets out of order** message, you are connecting with the service:

   If you have a MySQL client, log in with the standard CLI command:

   ```
   $ mysql -h 172.30.131.89 -u admin -p
   ```

   Example output

   ```
   Enter password:
   Welcome to the MariaDB monitor.  Commands end with ; or \g.
   ```

> MySQL [(none)]>

30.5. CONFIGURING INGRESS CLUSTER TRAFFIC ON AWS

OpenShift Container Platform provides methods for communicating from outside the cluster with services running in the cluster. This method uses load balancers on AWS, specifically a Network Load Balancer (NLB) or a Classic Load Balancer (CLB). Both types of load balancers can forward the client's IP address to the node, but a CLB requires proxy protocol support, which OpenShift Container Platform automatically enables.

There are two ways to configure an Ingress Controller to use an NLB:

1. By force replacing the Ingress Controller that is currently using a CLB. This deletes the **IngressController** object and an outage will occur while the new DNS records propagate and the NLB is being provisioned.

2. By editing an existing Ingress Controller that uses a CLB to use an NLB. This changes the load balancer without having to delete and recreate the **IngressController** object.

Both methods can be used to switch from an NLB to a CLB.

You can configure these load balancers on a new or existing AWS cluster.

30.5.1. Configuring Classic Load Balancer timeouts on AWS

OpenShift Container Platform provides a method for setting a custom timeout period for a specific route or Ingress Controller. Additionally, an AWS Classic Load Balancer (CLB) has its own timeout period with a default time of 60 seconds.

If the timeout period of the CLB is shorter than the route timeout or Ingress Controller timeout, the load balancer can prematurely terminate the connection. You can prevent this problem by increasing both the timeout period of the route and CLB.

30.5.1.1. Configuring route timeouts

You can configure the default timeouts for an existing route when you have services in need of a low timeout, which is required for Service Level Availability (SLA) purposes, or a high timeout, for cases with a slow back end.

Prerequisites

- You need a deployed Ingress Controller on a running cluster.

Procedure

1. Using the **oc annotate** command, add the timeout to the route:

    ```
    $ oc annotate route <route_name> \
        --overwrite haproxy.router.openshift.io/timeout=<timeout><time_unit>  ❶
    ```

    ❶ Supported time units are microseconds (us), milliseconds (ms), seconds (s), minutes (m), hours (h), or days (d).

The following example sets a timeout of two seconds on a route named **myroute**:

```
$ oc annotate route myroute --overwrite haproxy.router.openshift.io/timeout=2s
```

30.5.1.2. Configuring Classic Load Balancer timeouts

You can configure the default timeouts for a Classic Load Balancer (CLB) to extend idle connections.

Prerequisites

- You must have a deployed Ingress Controller on a running cluster.

Procedure

1. Set an AWS connection idle timeout of five minutes for the default **ingresscontroller** by running the following command:

```
$ oc -n openshift-ingress-operator patch ingresscontroller/default \
    --type=merge --patch='{"spec":{"endpointPublishingStrategy": \
    {"type":"LoadBalancerService", "loadBalancer": \
    {"scope":"External", "providerParameters":{"type":"AWS", "aws": \
    {"type":"Classic", "classicLoadBalancer": \
    {"connectionIdleTimeout":"5m"}}}}}}'
```

2. Optional: Restore the default value of the timeout by running the following command:

```
$ oc -n openshift-ingress-operator patch ingresscontroller/default \
    --type=merge --patch='{"spec":{"endpointPublishingStrategy": \
    {"loadBalancer":{"providerParameters":{"aws":{"classicLoadBalancer": \
    {"connectionIdleTimeout":null}}}}}}'
```

> **NOTE**
>
> You must specify the **scope** field when you change the connection timeout value unless the current scope is already set. When you set the **scope** field, you do not need to do so again if you restore the default timeout value.

30.5.2. Configuring ingress cluster traffic on AWS using a Network Load Balancer

OpenShift Container Platform provides methods for communicating from outside the cluster with services that run in the cluster. One such method uses a Network Load Balancer (NLB). You can configure an NLB on a new or existing AWS cluster.

30.5.2.1. Switching the Ingress Controller from using a Classic Load Balancer to a Network Load Balancer

You can switch the Ingress Controller that is using a Classic Load Balancer (CLB) to one that uses a Network Load Balancer (NLB) on AWS.

Switching between these load balancers will not delete the **IngressController** object.

<blockquote>

WARNING

This procedure might cause the following issues:

- An outage that can last several minutes due to new DNS records propagation, new load balancers provisioning, and other factors. IP addresses and canonical names of the Ingress Controller load balancer might change after applying this procedure.

- Leaked load balancer resources due to a change in the annotation of the service.

</blockquote>

Procedure

1. Modify the existing Ingress Controller that you want to switch to using an NLB. This example assumes that your default Ingress Controller has an **External** scope and no other customizations:

   Example ingresscontroller.yaml file

   ```
   apiVersion: operator.openshift.io/v1
   kind: IngressController
   metadata:
     creationTimestamp: null
     name: default
     namespace: openshift-ingress-operator
   spec:
     endpointPublishingStrategy:
       loadBalancer:
         scope: External
         providerParameters:
           type: AWS
           aws:
             type: NLB
       type: LoadBalancerService
   ```

   > **NOTE**
   >
   > If you do not specify a value for the **spec.endpointPublishingStrategy.loadBalancer.providerParameters.aws.type** field, the Ingress Controller uses the **spec.loadBalancer.platform.aws.type** value from the cluster **Ingress** configuration that was set during installation.

   TIP

   If your Ingress Controller has other customizations that you want to update, such as changing the domain, consider force replacing the Ingress Controller definition file instead.

2. Apply the changes to the Ingress Controller YAML file by running the command:

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

Expect several minutes of outages while the Ingress Controller updates.

30.5.2.2. Switching the Ingress Controller from using a Network Load Balancer to a Classic Load Balancer

You can switch the Ingress Controller that is using a Network Load Balancer (NLB) to one that uses a Classic Load Balancer (CLB) on AWS.

Switching between these load balancers will not delete the **IngressController** object.

> **WARNING**
>
> This procedure might cause an outage that can last several minutes due to new DNS records propagation, new load balancers provisioning, and other factors. IP addresses and canonical names of the Ingress Controller load balancer might change after applying this procedure.

Procedure

1. Modify the existing Ingress Controller that you want to switch to using a CLB. This example assumes that your default Ingress Controller has an **External** scope and no other customizations:

   Example ingresscontroller.yaml file

   ```yaml
   apiVersion: operator.openshift.io/v1
   kind: IngressController
   metadata:
     creationTimestamp: null
     name: default
     namespace: openshift-ingress-operator
   spec:
     endpointPublishingStrategy:
       loadBalancer:
         scope: External
         providerParameters:
           type: AWS
           aws:
             type: Classic
         type: LoadBalancerService
   ```

   > **NOTE**
   >
   > If you do not specify a value for the **spec.endpointPublishingStrategy.loadBalancer.providerParameters.aws.type** field, the Ingress Controller uses the **spec.loadBalancer.platform.aws.type** value from the cluster **Ingress** configuration that was set during installation.

TIP

If your Ingress Controller has other customizations that you want to update, such as changing the domain, consider force replacing the Ingress Controller definition file instead.

2. Apply the changes to the Ingress Controller YAML file by running the command:

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

Expect several minutes of outages while the Ingress Controller updates.

30.5.2.3. Replacing Ingress Controller Classic Load Balancer with Network Load Balancer

You can replace an Ingress Controller that is using a Classic Load Balancer (CLB) with one that uses a Network Load Balancer (NLB) on AWS.

> **WARNING**
>
> This procedure might cause the following issues:
>
> - An outage that can last several minutes due to new DNS records propagation, new load balancers provisioning, and other factors. IP addresses and canonical names of the Ingress Controller load balancer might change after applying this procedure.
>
> - Leaked load balancer resources due to a change in the annotation of the service.

Procedure

1. Create a file with a new default Ingress Controller. The following example assumes that your default Ingress Controller has an **External** scope and no other customizations:

   Example **ingresscontroller.yml** file

   ```
   apiVersion: operator.openshift.io/v1
   kind: IngressController
   metadata:
     creationTimestamp: null
     name: default
     namespace: openshift-ingress-operator
   spec:
     endpointPublishingStrategy:
       loadBalancer:
         scope: External
         providerParameters:
           type: AWS
           aws:
             type: NLB
       type: LoadBalancerService
   ```

If your default Ingress Controller has other customizations, ensure that you modify the file accordingly.

TIP

If your Ingress Controller has no other customizations and you are only updating the load balancer type, consider following the procedure detailed in "Switching the Ingress Controller from using a Classic Load Balancer to a Network Load Balancer".

2. Force replace the Ingress Controller YAML file:

   ```
   $ oc replace --force --wait -f ingresscontroller.yml
   ```

   Wait until the Ingress Controller is replaced. Expect several of minutes of outages.

30.5.2.4. Configuring an Ingress Controller Network Load Balancer on an existing AWS cluster

You can create an Ingress Controller backed by an AWS Network Load Balancer (NLB) on an existing cluster.

Prerequisites

- You must have an installed AWS cluster.

- **PlatformStatus** of the infrastructure resource must be AWS.

  - To verify that the **PlatformStatus** is AWS, run:

    ```
    $ oc get infrastructure/cluster -o jsonpath='{.status.platformStatus.type}'
    AWS
    ```

Procedure

Create an Ingress Controller backed by an AWS NLB on an existing cluster.

1. Create the Ingress Controller manifest:

   ```
   $ cat ingresscontroller-aws-nlb.yaml
   ```

   Example output

   ```
   apiVersion: operator.openshift.io/v1
   kind: IngressController
   metadata:
     name: $my_ingress_controller 1
     namespace: openshift-ingress-operator
   spec:
     domain: $my_unique_ingress_domain 2
     endpointPublishingStrategy:
       type: LoadBalancerService
       loadBalancer:
         scope: External 3
         providerParameters:
   ```

```
      type: AWS
      aws:
        type: NLB
```

1. Replace **$my_ingress_controller** with a unique name for the Ingress Controller.

2. Replace **$my_unique_ingress_domain** with a domain name that is unique among all Ingress Controllers in the cluster. This variable must be a subdomain of the DNS name **<clustername>.<domain>**.

3. You can replace **External** with **Internal** to use an internal NLB.

2. Create the resource in the cluster:

```
$ oc create -f ingresscontroller-aws-nlb.yaml
```
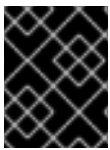
> **IMPORTANT**
>
> Before you can configure an Ingress Controller NLB on a new AWS cluster, you must complete the Creating the installation configuration file procedure.

30.5.2.5. Configuring an Ingress Controller Network Load Balancer on a new AWS cluster

You can create an Ingress Controller backed by an AWS Network Load Balancer (NLB) on a new cluster.

Prerequisites

- Create the **install-config.yaml** file and complete any modifications to it.

Procedure

Create an Ingress Controller backed by an AWS NLB on a new cluster.

1. Change to the directory that contains the installation program and create the manifests:

```
$ ./openshift-install create manifests --dir <installation_directory> ①
```

1. For **<installation_directory>**, specify the name of the directory that contains the **install-config.yaml** file for your cluster.

2. Create a file that is named **cluster-ingress-default-ingresscontroller.yaml** in the **<installation_directory>/manifests/** directory:

```
$ touch <installation_directory>/manifests/cluster-ingress-default-ingresscontroller.yaml ①
```

1. For **<installation_directory>**, specify the directory name that contains the **manifests/** directory for your cluster.

After creating the file, several network configuration files are in the **manifests/** directory, as shown:

```
$ ls <installation_directory>/manifests/cluster-ingress-default-ingresscontroller.yaml
```

Example output

> cluster-ingress-default-ingresscontroller.yaml

3. Open the **cluster-ingress-default-ingresscontroller.yaml** file in an editor and enter a custom resource (CR) that describes the Operator configuration you want:

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  creationTimestamp: null
  name: default
  namespace: openshift-ingress-operator
spec:
  endpointPublishingStrategy:
    loadBalancer:
      scope: External
      providerParameters:
        type: AWS
        aws:
          type: NLB
    type: LoadBalancerService
```

4. Save the **cluster-ingress-default-ingresscontroller.yaml** file and quit the text editor.

5. Optional: Back up the **manifests/cluster-ingress-default-ingresscontroller.yaml** file. The installation program deletes the **manifests/** directory when creating the cluster.

30.5.3. Additional resources

- Installing a cluster on AWS with network customizations .

- For more information on support for NLBs, see Network Load Balancer support on AWS .

- For more information on proxy protocol support for CLBs, see Configure proxy protocol support for your Classic Load Balancer

30.6. CONFIGURING INGRESS CLUSTER TRAFFIC FOR A SERVICE EXTERNAL IP

You can attach an external IP address to a service so that it is available to traffic outside the cluster. This is generally useful only for a cluster installed on bare metal hardware. The external network infrastructure must be configured correctly to route traffic to the service.

30.6.1. Prerequisites

- Your cluster is configured with ExternalIPs enabled. For more information, read Configuring ExternalIPs for services.
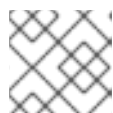
> **NOTE**
>
> Do not use the same ExternalIP for the egress IP.

30.6.2. Attaching an ExternalIP to a service

You can attach an ExternalIP to a service. If your cluster is configured to allocate an ExternalIP automatically, you might not need to manually attach an ExternalIP to the service.

Procedure

1. Optional: To confirm what IP address ranges are configured for use with ExternalIP, enter the following command:

   ```
   $ oc get networks.config cluster -o jsonpath='{.spec.externalIP}{"\n"}'
   ```

   If **autoAssignCIDRs** is set, OpenShift Container Platform automatically assigns an ExternalIP to a new **Service** object if the **spec.externalIPs** field is not specified.

2. Attach an ExternalIP to the service.

   a. If you are creating a new service, specify the **spec.externalIPs** field and provide an array of one or more valid IP addresses. For example:

   ```
   apiVersion: v1
   kind: Service
   metadata:
     name: svc-with-externalip
   spec:
     ...
     externalIPs:
     - 192.174.120.10
   ```

   b. If you are attaching an ExternalIP to an existing service, enter the following command. Replace **<name>** with the service name. Replace **<ip_address>** with a valid ExternalIP address. You can provide multiple IP addresses separated by commas.

   ```
   $ oc patch svc <name> -p \
    '{
      "spec": {
        "externalIPs": [ "<ip_address>" ]
      }
    }'
   ```

   For example:

   ```
   $ oc patch svc mysql-55-rhel7 -p '{"spec":{"externalIPs":["192.174.120.10"]}}'
   ```

   ### Example output

   ```
   "mysql-55-rhel7" patched
   ```

3. To confirm that an ExternalIP address is attached to the service, enter the following command. If you specified an ExternalIP for a new service, you must create the service first.

   ```
   $ oc get svc
   ```

   ### Example output

```
NAME          CLUSTER-IP     EXTERNAL-IP    PORT(S)   AGE
mysql-55-rhel7    172.30.131.89   192.174.120.10   3306/TCP   13m
```

30.6.3. Additional resources

- [Configuring ExternalIPs for services](#)

30.7. CONFIGURING INGRESS CLUSTER TRAFFIC USING A NODEPORT

OpenShift Container Platform provides methods for communicating from outside the cluster with services running in the cluster. This method uses a **NodePort**.

30.7.1. Using a NodePort to get traffic into the cluster

Use a **NodePort**-type **Service** resource to expose a service on a specific port on all nodes in the cluster. The port is specified in the **Service** resource's **.spec.ports[*].nodePort** field.

> **IMPORTANT**
>
> Using a node port requires additional port resources.

A **NodePort** exposes the service on a static port on the node's IP address. **NodePort**s are in the **30000** to **32767** range by default, which means a **NodePort** is unlikely to match a service's intended port. For example, port **8080** may be exposed as port **31020** on the node.

The administrator must ensure the external IP addresses are routed to the nodes.

NodePorts and external IPs are independent and both can be used concurrently.

> **NOTE**
>
> The procedures in this section require prerequisites performed by the cluster administrator.

30.7.2. Prerequisites

Before starting the following procedures, the administrator must:

- Set up the external port to the cluster networking environment so that requests can reach the cluster.

- Make sure there is at least one user with cluster admin role. To add this role to a user, run the following command:

  ```
  $ oc adm policy add-cluster-role-to-user cluster-admin <user_name>
  ```

- Have an OpenShift Container Platform cluster with at least one master and at least one node and a system outside the cluster that has network access to the cluster. This procedure assumes that the external system is on the same subnet as the cluster. The additional networking required for external systems on a different subnet is out-of-scope for this topic.

30.7.3. Creating a project and service

If the project and service that you want to expose do not exist, first create the project, then the service.

If the project and service already exist, skip to the procedure on exposing the service to create a route.

Prerequisites

- Install the **oc** CLI and log in as a cluster administrator.

Procedure

1. Create a new project for your service by running the **oc new-project** command:

   ```
   $ oc new-project myproject
   ```

2. Use the **oc new-app** command to create your service:

   ```
   $ oc new-app nodejs:12~https://github.com/sclorg/nodejs-ex.git
   ```

3. To verify that the service was created, run the following command:

   ```
   $ oc get svc -n myproject
   ```

   Example output

   ```
   NAME       TYPE       CLUSTER-IP      EXTERNAL-IP  PORT(S)   AGE
   nodejs-ex  ClusterIP  172.30.197.157  <none>       8080/TCP  70s
   ```

   By default, the new service does not have an external IP address.

30.7.4. Exposing the service by creating a route

You can expose the service as a route by using the **oc expose** command.

Procedure

To expose the service:

1. Log in to OpenShift Container Platform.

2. Log in to the project where the service you want to expose is located:

   ```
   $ oc project myproject
   ```

3. To expose a node port for the application, modify the custom resource definition (CRD) of a service by entering the following command:

   ```
   $ oc edit svc <service_name>
   ```

   Example output

   ```
   spec:
   ```

```
    ports:
    - name: 8443-tcp
      nodePort: 30327  1
      port: 8443
      protocol: TCP
      targetPort: 8443
    sessionAffinity: None
    type: NodePort  2
```

1   Optional: Specify the node port range for the application. By default, OpenShift Container Platform selects an available port in the **30000-32767** range.

2   Define the service type.

4. Optional: To confirm the service is available with a node port exposed, enter the following command:

   ```
   $ oc get svc -n myproject
   ```

   Example output

   ```
   NAME             TYPE        CLUSTER-IP      EXTERNAL-IP  PORT(S)        AGE
   nodejs-ex        ClusterIP   172.30.217.127  <none>       3306/TCP       9m44s
   nodejs-ex-ingress  NodePort  172.30.107.72   <none>       3306:31345/TCP  39s
   ```

5. Optional: To remove the service created automatically by the **oc new-app** command, enter the following command:

   ```
   $ oc delete svc nodejs-ex
   ```

Verification

- To check that the service node port is updated with a port in the **30000-32767** range, enter the following command:

  ```
  $ oc get svc
  ```

  In the following example output, the updated port is **30327**:

  Example output

  ```
  NAME   TYPE      CLUSTER-IP    EXTERNAL-IP  PORT(S)        AGE
  httpd  NodePort  172.xx.xx.xx  <none>       8443:30327/TCP  109s
  ```

30.7.5. Additional resources

- [Configuring the node port service range](#)

30.8. CONFIGURING INGRESS CLUSTER TRAFFIC USING LOAD BALANCER ALLOWED SOURCE RANGES

You can specify a list of IP address ranges for the **IngressController**. This restricts access to the load balancer service when the **endpointPublishingStrategy** is **LoadBalancerService**.

30.8.1. Configuring load balancer allowed source ranges

You can enable and configure the **spec.endpointPublishingStrategy.loadBalancer.allowedSourceRanges** field. By configuring load balancer allowed source ranges, you can limit the access to the load balancer for the Ingress Controller to a specified list of IP address ranges. The Ingress Operator reconciles the load balancer Service and sets the **spec.loadBalancerSourceRanges** field based on **AllowedSourceRanges**.
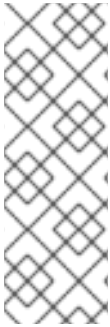
> **NOTE**
>
> If you have already set the **spec.loadBalancerSourceRanges** field or the load balancer service anotation **service.beta.kubernetes.io/load-balancer-source-ranges** in a previous version of OpenShift Container Platform, Ingress Controller starts reporting **Progressing=True** after an upgrade. To fix this, set **AllowedSourceRanges** that overwrites the **spec.loadBalancerSourceRanges** field and clears the **service.beta.kubernetes.io/load-balancer-source-ranges** annotation. Ingress Controller starts reporting **Progressing=False** again.

Prerequisites

- You have a deployed Ingress Controller on a running cluster.

Procedure

- Set the allowed source ranges API for the Ingress Controller by running the following command:

  ```
  $ oc -n openshift-ingress-operator patch ingresscontroller/default \
      --type=merge --patch='{"spec":{"endpointPublishingStrategy": \
      {"loadBalancer":{"allowedSourceRanges":["0.0.0.0/0"]}}}}'
  ```
  1

  1   The example value **0.0.0.0/0** specifies the allowed source range.

30.8.2. Migrating to load balancer allowed source ranges

If you have already set the annotation **service.beta.kubernetes.io/load-balancer-source-ranges**, you can migrate to load balancer allowed source ranges. When you set the **AllowedSourceRanges**, the Ingress Controller sets the **spec.loadBalancerSourceRanges** field based on the **AllowedSourceRanges** value and unsets the **service.beta.kubernetes.io/load-balancer-source-ranges** annotation.

> **NOTE**
>
> If you have already set the **spec.loadBalancerSourceRanges** field or the load balancer service anotation **service.beta.kubernetes.io/load-balancer-source-ranges** in a previous version of OpenShift Container Platform, the Ingress Controller starts reporting **Progressing=True** after an upgrade. To fix this, set **AllowedSourceRanges** that overwrites the **spec.loadBalancerSourceRanges** field and clears the **service.beta.kubernetes.io/load-balancer-source-ranges** annotation. The Ingress Controller starts reporting **Progressing=False** again.

Prerequisites

- You have set the **service.beta.kubernetes.io/load-balancer-source-ranges** annotation.

Procedure

1. Ensure that the **service.beta.kubernetes.io/load-balancer-source-ranges** is set:

```
$ oc get svc router-default -n openshift-ingress -o yaml
```

Example output

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.beta.kubernetes.io/load-balancer-source-ranges: 192.168.0.1/32
```

2. Ensure that the **spec.loadBalancerSourceRanges** field is unset:

```
$ oc get svc router-default -n openshift-ingress -o yaml
```

Example output

```
...
spec:
  loadBalancerSourceRanges:
  - 0.0.0.0/0
...
```

3. Update your cluster to OpenShift Container Platform 4.13.

4. Set the allowed source ranges API for the **ingresscontroller** by running the following command:

```
$ oc -n openshift-ingress-operator patch ingresscontroller/default \
    --type=merge --patch='{"spec":{"endpointPublishingStrategy": \
    {"loadBalancer":{"allowedSourceRanges":["0.0.0.0/0"]}}}}'  1
```

1 The example value **0.0.0.0/0** specifies the allowed source range.

30.8.3. Additional resources

- Updating your cluster

CHAPTER 31. KUBERNETES NMSTATE

31.1. ABOUT THE KUBERNETES NMSTATE OPERATOR

The Kubernetes NMState Operator provides a Kubernetes API for performing state-driven network configuration across the OpenShift Container Platform cluster's nodes with NMState. The Kubernetes NMState Operator provides users with functionality to configure various network interface types, DNS, and routing on cluster nodes. Additionally, the daemons on the cluster nodes periodically report on the state of each node's network interfaces to the API server.

> **IMPORTANT**
>
> Red Hat supports the Kubernetes NMState Operator in production environments on bare-metal, IBM Power, IBM Z, IBM® LinuxONE, VMware vSphere, and OpenStack installations.

Before you can use NMState with OpenShift Container Platform, you must install the Kubernetes NMState Operator.

> **NOTE**
>
> The Kubernetes NMState Operator updates the network configuration of a secondary NIC. It cannot update the network configuration of the primary NIC or the **br-ex** bridge.

OpenShift Container Platform uses **nmstate** to report on and configure the state of the node network. This makes it possible to modify the network policy configuration, such as by creating a Linux bridge on all nodes, by applying a single configuration manifest to the cluster.

Node networking is monitored and updated by the following objects:

NodeNetworkState

Reports the state of the network on that node.

NodeNetworkConfigurationPolicy

Describes the requested network configuration on nodes. You update the node network configuration, including adding and removing interfaces, by applying a **NodeNetworkConfigurationPolicy** manifest to the cluster.

NodeNetworkConfigurationEnactment

Reports the network policies enacted upon each node.

31.1.1. Installing the Kubernetes NMState Operator

You can install the Kubernetes NMState Operator by using the web console or the CLI.

31.1.1.1. Installing the Kubernetes NMState Operator using the web console

You can install the Kubernetes NMState Operator by using the web console. After it is installed, the Operator can deploy the NMState State Controller as a daemon set across all of the cluster nodes.

Prerequisites

- You are logged in as a user with **cluster-admin** privileges.

Procedure

1. Select **Operators → OperatorHub**.

2. In the search field below **All Items**, enter **nmstate** and click **Enter** to search for the Kubernetes NMState Operator.

3. Click on the Kubernetes NMState Operator search result.

4. Click on **Install** to open the **Install Operator** window.

5. Click **Install** to install the Operator.

6. After the Operator finishes installing, click **View Operator**.

7. Under **Provided APIs**, click **Create Instance** to open the dialog box for creating an instance of **kubernetes-nmstate**.

8. In the **Name** field of the dialog box, ensure the name of the instance is **nmstate.**

> **NOTE**
>
> The name restriction is a known issue. The instance is a singleton for the entire cluster.

9. Accept the default settings and click **Create** to create the instance.

Summary

Once complete, the Operator has deployed the NMState State Controller as a daemon set across all of the cluster nodes.

31.1.1.2. Installing the Kubernetes NMState Operator using the CLI

You can install the Kubernetes NMState Operator by using the OpenShift CLI (**oc**). After it is installed, the Operator can deploy the NMState State Controller as a daemon set across all of the cluster nodes.

Prerequisites

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

- You are logged in as a user with **cluster-admin** privileges.

Procedure

1. Create the **nmstate** Operator namespace:

```
$ cat << EOF | oc apply -f -
apiVersion: v1
kind: Namespace
metadata:
  labels:
    kubernetes.io/metadata.name: openshift-nmstate
    name: openshift-nmstate
  name: openshift-nmstate
spec:
```

```
  finalizers:
  - kubernetes
EOF
```

2. Create the **OperatorGroup**:

```
$ cat << EOF | oc apply -f -
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  annotations:
    olm.providedAPIs: NMState.v1.nmstate.io
  name: openshift-nmstate
  namespace: openshift-nmstate
spec:
  targetNamespaces:
  - openshift-nmstate
EOF
```

3. Subscribe to the **nmstate** Operator:

```
$ cat << EOF| oc apply -f -
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  labels:
    operators.coreos.com/kubernetes-nmstate-operator.openshift-nmstate: ""
  name: kubernetes-nmstate-operator
  namespace: openshift-nmstate
spec:
  channel: stable
  installPlanApproval: Automatic
  name: kubernetes-nmstate-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
EOF
```

4. Create instance of the **nmstate** operator:

```
$ cat << EOF | oc apply -f -
apiVersion: nmstate.io/v1
kind: NMState
metadata:
  name: nmstate
EOF
```

Verification

- Confirm that the deployment for the **nmstate** operator is running:

```
oc get clusterserviceversion -n openshift-nmstate \
 -o custom-columns=Name:.metadata.name,Phase:.status.phase
```

Example output

```
Name                                           Phase
kubernetes-nmstate-operator.4.13.0-202210210157    Succeeded
```

31.2. OBSERVING AND UPDATING THE NODE NETWORK STATE AND CONFIGURATION

31.2.1. Viewing the network state of a node

Node network state is the network configuration for all nodes in the cluster. A **NodeNetworkState** object exists on every node in the cluster. This object is periodically updated and captures the state of the network for that node.

Procedure

1. List all the **NodeNetworkState** objects in the cluster:

   ```
   $ oc get nns
   ```

2. Inspect a **NodeNetworkState** object to view the network on that node. The output in this example has been redacted for clarity:

   ```
   $ oc get nns node01 -o yaml
   ```

   Example output

   ```
   apiVersion: nmstate.io/v1
   kind: NodeNetworkState
   metadata:
     name: node01 1
   status:
     currentState: 2
       dns-resolver:
   # ...
       interfaces:
   # ...
       route-rules:
   # ...
       routes:
   # ...
     lastSuccessfulUpdateTime: "2020-01-31T12:14:00Z" 3
   ```

   1 The name of the **NodeNetworkState** object is taken from the node.

   2 The **currentState** contains the complete network configuration for the node, including DNS, interfaces, and routes.

   3 Timestamp of the last successful update. This is updated periodically as long as the node is reachable and can be used to evalute the freshness of the report.

31.2.2. Managing policy by using the CLI

31.2.2.1. Creating an interface on nodes

Create an interface on nodes in the cluster by applying a **NodeNetworkConfigurationPolicy** manifest to the cluster. The manifest details the requested configuration for the interface.

By default, the manifest applies to all nodes in the cluster. To add the interface to specific nodes, add the **spec: nodeSelector** parameter and the appropriate **<key>:<value>** for your node selector.

You can configure multiple nmstate-enabled nodes concurrently. The configuration applies to 50% of the nodes in parallel. This strategy prevents the entire cluster from being unavailable if the network connection fails. To apply the policy configuration in parallel to a specific portion of the cluster, use the **maxUnavailable** field.

Procedure

1. Create the **NodeNetworkConfigurationPolicy** manifest. The following example configures a Linux bridge on all worker nodes and configures the DNS resolver:

   ```
   apiVersion: nmstate.io/v1
   kind: NodeNetworkConfigurationPolicy
   metadata:
     name: br1-eth1-policy ❶
   spec:
     nodeSelector: ❷
       node-role.kubernetes.io/worker: "" ❸
     maxUnavailable: 3 ❹
     desiredState:
       interfaces:
         - name: br1
           description: Linux bridge with eth1 as a port ❺
           type: linux-bridge
           state: up
           ipv4:
             dhcp: true
             enabled: true
             auto-dns: false
           bridge:
             options:
               stp:
                 enabled: false
             port:
               - name: eth1
       dns-resolver: ❻
         config:
           search:
           - example.com
           - example.org
           server:
           - 8.8.8.8
   ```

   ❶ Name of the policy.

   ❷ Optional: If you do not include the **nodeSelector** parameter, the policy applies to all nodes in the cluster.

3 This example uses the **node-role.kubernetes.io/worker: ""** node selector to select all worker nodes in the cluster.

4 Optional: Specifies the maximum number of nmstate-enabled nodes that the policy configuration can be applied to concurrently. This parameter can be set to either a percentage value (string), for example, **"10%"**, or an absolute value (number), such as **3**.

5 Optional: Human-readable description for the interface.

6 Optional: Specifies the search and server settings for the DNS server.

2. Create the node network policy:

```
$ oc apply -f br1-eth1-policy.yaml 1
```

1 File name of the node network configuration policy manifest.

Additional resources

- Example for creating multiple interfaces in the same policy

- Examples of different IP management methods in policies

31.2.3. Confirming node network policy updates on nodes

A **NodeNetworkConfigurationPolicy** manifest describes your requested network configuration for nodes in the cluster. The node network policy includes your requested network configuration and the status of execution of the policy on the cluster as a whole.

When you apply a node network policy, a **NodeNetworkConfigurationEnactment** object is created for every node in the cluster. The node network configuration enactment is a read-only object that represents the status of execution of the policy on that node. If the policy fails to be applied on the node, the enactment for that node includes a traceback for troubleshooting.

Procedure

1. To confirm that a policy has been applied to the cluster, list the policies and their status:

```
$ oc get nncp
```

2. Optional: If a policy is taking longer than expected to successfully configure, you can inspect the requested state and status conditions of a particular policy:

```
$ oc get nncp <policy> -o yaml
```

3. Optional: If a policy is taking longer than expected to successfully configure on all nodes, you can list the status of the enactments on the cluster:

```
$ oc get nnce
```

4. Optional: To view the configuration of a particular enactment, including any error reporting for a failed configuration:

```
$ oc get nnce <node>.<policy> -o yaml
```

31.2.4. Removing an interface from nodes

You can remove an interface from one or more nodes in the cluster by editing the **NodeNetworkConfigurationPolicy** object and setting the **state** of the interface to **absent**.

Removing an interface from a node does not automatically restore the node network configuration to a previous state. If you want to restore the previous state, you will need to define that node network configuration in the policy.

If you remove a bridge or bonding interface, any node NICs in the cluster that were previously attached or subordinate to that bridge or bonding interface are placed in a **down** state and become unreachable. To avoid losing connectivity, configure the node NIC in the same policy so that it has a status of **up** and either DHCP or a static IP address.

> **NOTE**
>
> Deleting the node network policy that added an interface does not change the configuration of the policy on the node. Although a **NodeNetworkConfigurationPolicy** is an object in the cluster, it only represents the requested configuration. Similarly, removing an interface does not delete the policy.

Procedure

1. Update the **NodeNetworkConfigurationPolicy** manifest used to create the interface. The following example removes a Linux bridge and configures the **eth1** NIC with DHCP to avoid losing connectivity:

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: <br1-eth1-policy> 1
spec:
  nodeSelector: 2
    node-role.kubernetes.io/worker: "" 3
  desiredState:
    interfaces:
    - name: br1
      type: linux-bridge
      state: absent 4
    - name: eth1 5
      type: ethernet 6
      state: up 7
      ipv4:
        dhcp: true 8
        enabled: true 9
```

1   Name of the policy.

2   Optional: If you do not include the **nodeSelector** parameter, the policy applies to all nodes in the cluster.

3 This example uses the **node-role.kubernetes.io/worker: ""** node selector to select all worker nodes in the cluster.

4 Changing the state to **absent** removes the interface.

5 The name of the interface that is to be unattached from the bridge interface.

6 The type of interface. This example creates an Ethernet networking interface.

7 The requested state for the interface.

8 Optional: If you do not use **dhcp**, you can either set a static IP or leave the interface without an IP address.

9 Enables **ipv4** in this example.

2. Update the policy on the node and remove the interface:

```
$ oc apply -f <br1-eth1-policy.yaml> 1
```

1 File name of the policy manifest.

31.2.5. Example policy configurations for different interfaces

The following examples show different **NodeNetworkConfigurationPolicy** manifest configurations.

For best performance, consider the following factors when applying a policy:

- When you need to apply a policy to more than one node, create a **NodeNetworkConfigurationPolicy** manifest for each target node. Scoping a policy to a single node reduces the overall length of time for the Kubernetes NMState Operator to apply the policies.
  In contrast, if a single policy includes configurations for several nodes, the Kubernetes NMState Operator applies the policy to each node in sequence, which increases the overall length of time for policy application.

- All related network configurations should be specified in a single policy.
  When a node restarts, the Kubernetes NMState Operator cannot control the order in which policies are applied. Therefore, the Kubernetes NMState Operator might apply interdependent policies in a sequence that results in a degraded network object.

31.2.5.1. Example: Linux bridge interface node network configuration policy

Create a Linux bridge interface on nodes in the cluster by applying a **NodeNetworkConfigurationPolicy** manifest to the cluster.

The following YAML file is an example of a manifest for a Linux bridge interface. It includes samples values that you must replace with your own information.

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: br1-eth1-policy 1
```

```
  spec:
   nodeSelector: 2
    kubernetes.io/hostname: <node01> 3
   desiredState:
    interfaces:
     - name: br1 4
       description: Linux bridge with eth1 as a port 5
       type: linux-bridge 6
       state: up 7
       ipv4:
        dhcp: true 8
        enabled: true 9
       bridge:
        options:
         stp:
           enabled: false 10
        port:
         - name: eth1 11
```

[1] Name of the policy.

[2] Optional: If you do not include the **nodeSelector** parameter, the policy applies to all nodes in the cluster.

[3] This example uses a **hostname** node selector.

[4] Name of the interface.

[5] Optional: Human-readable description of the interface.

[6] The type of interface. This example creates a bridge.

[7] The requested state for the interface after creation.

[8] Optional: If you do not use **dhcp**, you can either set a static IP or leave the interface without an IP address.

[9] Enables **ipv4** in this example.

[10] Disables **stp** in this example.

[11] The node NIC to which the bridge attaches.

31.2.5.2. Example: VLAN interface node network configuration policy

Create a VLAN interface on nodes in the cluster by applying a **NodeNetworkConfigurationPolicy** manifest to the cluster.

> **NOTE**
>
> Define all related configurations for the VLAN interface of a node in a single **NodeNetworkConfigurationPolicy** manifest. For example, define the VLAN interface for a node and the related routes for the VLAN interface in the same **NodeNetworkConfigurationPolicy** manifest.
>
> When a node restarts, the Kubernetes NMState Operator cannot control the order in which policies are applied. Therefore, if you use separate policies for related network configurations, the Kubernetes NMState Operator might apply these policies in a sequence that results in a degraded network object.

The following YAML file is an example of a manifest for a VLAN interface. It includes samples values that you must replace with your own information.

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: vlan-eth1-policy 1
spec:
  nodeSelector: 2
    kubernetes.io/hostname: <node01> 3
  desiredState:
    interfaces:
    - name: eth1.102 4
      description: VLAN using eth1 5
      type: vlan 6
      state: up 7
      vlan:
        base-iface: eth1 8
        id: 102 9
```

1	Name of the policy.
2	Optional: If you do not include the **nodeSelector** parameter, the policy applies to all nodes in the cluster.
3	This example uses a **hostname** node selector.
4	Name of the interface. When deploying on bare metal, only the **<interface_name>.<vlan_number>** VLAN format is supported.
5	Optional: Human-readable description of the interface.
6	The type of interface. This example creates a VLAN.
7	The requested state for the interface after creation.
8	The node NIC to which the VLAN is attached.
9	The VLAN tag.

31.2.5.3. Example: Bond interface node network configuration policy

Create a bond interface on nodes in the cluster by applying a **NodeNetworkConfigurationPolicy** manifest to the cluster.

> **NOTE**
>
> OpenShift Container Platform only supports the following bond modes:
>
> - mode=1 active-backup
> - mode=2 balance-xor
> - mode=4 802.3ad
> - mode=5 balance-tlb
> - mode=6 balance-alb

The following YAML file is an example of a manifest for a bond interface. It includes samples values that you must replace with your own information.

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: bond0-eth1-eth2-policy 1
spec:
  nodeSelector: 2
    kubernetes.io/hostname: <node01> 3
  desiredState:
    interfaces:
    - name: bond0 4
      description: Bond with ports eth1 and eth2 5
      type: bond 6
      state: up 7
      ipv4:
        dhcp: true 8
        enabled: true 9
      link-aggregation:
        mode: active-backup 10
        options:
          miimon: '140' 11
        port: 12
        - eth1
        - eth2
      mtu: 1450 13
```

1	Name of the policy.

2	Optional: If you do not include the **nodeSelector** parameter, the policy applies to all nodes in the cluster.

3	This example uses a **hostname** node selector.

4	Name of the interface.

5   Optional: Human-readable description of the interface.

6   The type of interface. This example creates a bond.

7   The requested state for the interface after creation.

8   Optional: If you do not use **dhcp**, you can either set a static IP or leave the interface without an IP address.

9   Enables **ipv4** in this example.

10   The driver mode for the bond. This example uses an active backup mode.

11   Optional: This example uses miimon to inspect the bond link every 140ms.

12   The subordinate node NICs in the bond.

13   Optional: The maximum transmission unit (MTU) for the bond. If not specified, this value is set to **1500** by default.

31.2.5.4. Example: Ethernet interface node network configuration policy

Configure an Ethernet interface on nodes in the cluster by applying a **NodeNetworkConfigurationPolicy** manifest to the cluster.

The following YAML file is an example of a manifest for an Ethernet interface. It includes sample values that you must replace with your own information.

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
 name: eth1-policy 1
spec:
 nodeSelector: 2
   kubernetes.io/hostname: <node01> 3
 desiredState:
  interfaces:
  - name: eth1 4
    description: Configuring eth1 on node01 5
    type: ethernet 6
    state: up 7
    ipv4:
      dhcp: true 8
      enabled: true 9
```

1   Name of the policy.

2   Optional: If you do not include the **nodeSelector** parameter, the policy applies to all nodes in the cluster.

3   This example uses a **hostname** node selector.

4   Name of the interface.

5   Optional: Human-readable description of the interface.

6   The type of interface. This example creates an Ethernet networking interface.

7   The requested state for the interface after creation.

8   Optional: If you do not use **dhcp**, you can either set a static IP or leave the interface without an IP address.

9   Enables **ipv4** in this example.

31.2.5.5. Example: Multiple interfaces in the same node network configuration policy

You can create multiple interfaces in the same node network configuration policy. These interfaces can reference each other, allowing you to build and deploy a network configuration by using a single policy manifest.

The following example YAML file creates a bond that is named **bond10** across two NICs and VLAN that is named **bond10.103** that connects to the bond.

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: bond-vlan 1
spec:
  nodeSelector: 2
    kubernetes.io/hostname: <node01> 3
  desiredState:
    interfaces:
    - name: bond10 4
      description: Bonding eth2 and eth3 5
      type: bond 6
      state: up 7
      link-aggregation:
        mode: balance-rr 8
        options:
          miimon: '140' 9
        port: 10
        - eth2
        - eth3
    - name: bond10.103 11
      description: vlan using bond10 12
      type: vlan 13
      state: up 14
      vlan:
        base-iface: bond10 15
        id: 103 16
      ipv4:
        dhcp: true 17
        enabled: true 18
```

1   Name of the policy.

2   Optional: If you do not include the **nodeSelector** parameter, the policy applies to all nodes in the cluster.

3   This example uses **hostname** node selector.

4 **11** Name of the interface.

5 **12** Optional: Human-readable description of the interface.

6 **13** The type of interface.

7 **14** The requested state for the interface after creation.

8   The driver mode for the bond.

9   Optional: This example uses miimon to inspect the bond link every 140ms.

10   The subordinate node NICs in the bond.

15   The node NIC to which the VLAN is attached.

16   The VLAN tag.

17   Optional: If you do not use dhcp, you can either set a static IP or leave the interface without an IP address.

18   Enables ipv4 in this example.

31.2.6. Capturing the static IP of a NIC attached to a bridge

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

31.2.6.1. Example: Linux bridge interface node network configuration policy to inherit static IP address from the NIC attached to the bridge

Create a Linux bridge interface on nodes in the cluster and transfer the static IP configuration of the NIC to the bridge by applying a single **NodeNetworkConfigurationPolicy** manifest to the cluster.

The following YAML file is an example of a manifest for a Linux bridge interface. It includes sample values that you must replace with your own information.

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: br1-eth1-copy-ipv4-policy 1
```

```
spec:
  nodeSelector: 2
    node-role.kubernetes.io/worker: ""
  capture:
    eth1-nic: interfaces.name=="eth1" 3
    eth1-routes: routes.running.next-hop-interface=="eth1"
    br1-routes: capture.eth1-routes | routes.running.next-hop-interface := "br1"
  desiredState:
    interfaces:
      - name: br1
        description: Linux bridge with eth1 as a port
        type: linux-bridge 4
        state: up
        ipv4: "{{ capture.eth1-nic.interfaces.0.ipv4 }}" 5
        bridge:
          options:
            stp:
              enabled: false
          port:
            - name: eth1 6
    routes:
      config: "{{ capture.br1-routes.routes.running }}"
```

[1] The name of the policy.

[2] Optional: If you do not include the **nodeSelector** parameter, the policy applies to all nodes in the cluster. This example uses the **node-role.kubernetes.io/worker: ""** node selector to select all worker nodes in the cluster.

[3] The reference to the node NIC to which the bridge attaches.

[4] The type of interface. This example creates a bridge.

[5] The IP address of the bridge interface. This value matches the IP address of the NIC which is referenced by the **spec.capture.eth1-nic** entry.

[6] The node NIC to which the bridge attaches.

Additional resources

- The NMPolicy project – Policy syntax

31.2.7. Examples: IP management

The following example configuration snippets demonstrate different methods of IP management.

These examples use the **ethernet** interface type to simplify the example while showing the related context in the policy configuration. These IP management examples can be used with the other interface types.

31.2.7.1. Static

The following snippet statically configures an IP address on the Ethernet interface:

```
# ...
   interfaces:
   - name: eth1
     description: static IP on eth1
     type: ethernet
     state: up
     ipv4:
       dhcp: false
       address:
       - ip: 192.168.122.250 1
         prefix-length: 24
       enabled: true
# ...
```

[1]   Replace this value with the static IP address for the interface.

31.2.7.2. No IP address

The following snippet ensures that the interface has no IP address:

```
# ...
   interfaces:
   - name: eth1
     description: No IP on eth1
     type: ethernet
     state: up
     ipv4:
       enabled: false
# ...
```

31.2.7.3. Dynamic host configuration

The following snippet configures an Ethernet interface that uses a dynamic IP address, gateway address, and DNS:

```
# ...
   interfaces:
   - name: eth1
     description: DHCP on eth1
     type: ethernet
     state: up
     ipv4:
       dhcp: true
       enabled: true
# ...
```

The following snippet configures an Ethernet interface that uses a dynamic IP address but does not use a dynamic gateway address or DNS:

```
# ...
   interfaces:
   - name: eth1
     description: DHCP without gateway or DNS on eth1
```

```
    type: ethernet
    state: up
    ipv4:
      dhcp: true
      auto-gateway: false
      auto-dns: false
      enabled: true
# ...
```

31.2.7.4. DNS

Setting the DNS configuration is analagous to modifying the **/etc/resolv.conf** file. The following snippet sets the DNS configuration on the host.

```
# ...
    interfaces: ❶
      ...
      ipv4:
        ...
        auto-dns: false
        ...
    dns-resolver:
      config:
        search:
        - example.com
        - example.org
        server:
        - 8.8.8.8
# ...
```

❶ You must configure an interface with **auto-dns: false** or you must use static IP configuration on an interface in order for Kubernetes NMState to store custom DNS settings.

> **IMPORTANT**
>
> You cannot use **br-ex**, an OVNKubernetes-managed Open vSwitch bridge, as the interface when configuring DNS resolvers.

31.2.7.5. Static routing

The following snippet configures a static route and a static IP on interface **eth1**.

```
# ...
    interfaces:
    - name: eth1
      description: Static routing on eth1
      type: ethernet
      state: up
      ipv4:
        dhcp: false
        address:
        - ip: 192.0.2.251 ❶
          prefix-length: 24
```

```
      enabled: true
    routes:
      config:
      - destination: 198.51.100.0/24
        metric: 150
        next-hop-address: 192.0.2.1  2
        next-hop-interface: eth1
        table-id: 254
    # ...
```

1   The static IP address for the Ethernet interface.

2   Next hop address for the node traffic. This must be in the same subnet as the IP address set for the Ethernet interface.

31.3. TROUBLESHOOTING NODE NETWORK CONFIGURATION

If the node network configuration encounters an issue, the policy is automatically rolled back and the enactments report failure. This includes issues such as:

- The configuration fails to be applied on the host.

- The host loses connection to the default gateway.

- The host loses connection to the API server.

31.3.1. Troubleshooting an incorrect node network configuration policy configuration

You can apply changes to the node network configuration across your entire cluster by applying a node network configuration policy. If you apply an incorrect configuration, you can use the following example to troubleshoot and correct the failed node network policy.

In this example, a Linux bridge policy is applied to an example cluster that has three control plane nodes and three compute nodes. The policy fails to be applied because it references an incorrect interface. To find the error, investigate the available NMState resources. You can then update the policy with the correct configuration.

Procedure

1. Create a policy and apply it to your cluster. The following example creates a simple bridge on the **ens01** interface:

   ```
   apiVersion: nmstate.io/v1
   kind: NodeNetworkConfigurationPolicy
   metadata:
     name: ens01-bridge-testfail
   spec:
     desiredState:
       interfaces:
         - name: br1
           description: Linux bridge with the wrong port
           type: linux-bridge
           state: up
   ```

```
    ipv4:
      dhcp: true
      enabled: true
    bridge:
      options:
        stp:
          enabled: false
      port:
        - name: ens01
```

```
$ oc apply -f ens01-bridge-testfail.yaml
```

Example output

```
nodenetworkconfigurationpolicy.nmstate.io/ens01-bridge-testfail created
```

2. Verify the status of the policy by running the following command:

```
$ oc get nncp
```

The output shows that the policy failed:

Example output

```
NAME                STATUS
ens01-bridge-testfail   FailedToConfigure
```

However, the policy status alone does not indicate if it failed on all nodes or a subset of nodes.

3. List the node network configuration enactments to see if the policy was successful on any of the nodes. If the policy failed for only a subset of nodes, it suggests that the problem is with a specific node configuration. If the policy failed on all nodes, it suggests that the problem is with the policy.

```
$ oc get nnce
```

The output shows that the policy failed on all nodes:

Example output

```
NAME                                  STATUS
control-plane-1.ens01-bridge-testfail      FailedToConfigure
control-plane-2.ens01-bridge-testfail      FailedToConfigure
control-plane-3.ens01-bridge-testfail      FailedToConfigure
compute-1.ens01-bridge-testfail         FailedToConfigure
compute-2.ens01-bridge-testfail         FailedToConfigure
compute-3.ens01-bridge-testfail         FailedToConfigure
```

4. View one of the failed enactments and look at the traceback. The following command uses the output tool **jsonpath** to filter the output:

```
$ oc get nnce compute-1.ens01-bridge-testfail -o jsonpath='{.status.conditions[?
(@.type=="Failing")].message}'
```

This command returns a large traceback that has been edited for brevity:

Example output

```
error reconciling NodeNetworkConfigurationPolicy at desired state apply: , failed to execute
nmstatectl set --no-commit --timeout 480: 'exit status 1' "
...
libnmstate.error.NmstateVerificationError:
desired
=======
---
name: br1
type: linux-bridge
state: up
bridge:
  options:
    group-forward-mask: 0
    mac-ageing-time: 300
    multicast-snooping: true
    stp:
      enabled: false
      forward-delay: 15
      hello-time: 2
      max-age: 20
      priority: 32768
  port:
  - name: ens01
description: Linux bridge with the wrong port
ipv4:
  address: []
  auto-dns: true
  auto-gateway: true
  auto-routes: true
  dhcp: true
  enabled: true
ipv6:
  enabled: false
mac-address: 01-23-45-67-89-AB
mtu: 1500

current
=======
---
name: br1
type: linux-bridge
state: up
bridge:
  options:
    group-forward-mask: 0
    mac-ageing-time: 300
    multicast-snooping: true
    stp:
      enabled: false
      forward-delay: 15
      hello-time: 2
      max-age: 20
```

```
      priority: 32768
    port: []
  description: Linux bridge with the wrong port
  ipv4:
    address: []
    auto-dns: true
    auto-gateway: true
    auto-routes: true
    dhcp: true
    enabled: true
  ipv6:
    enabled: false
  mac-address: 01-23-45-67-89-AB
  mtu: 1500

difference
==========
--- desired
+++ current
@@ -13,8 +13,7 @@
      hello-time: 2
      max-age: 20
      priority: 32768
-  port:
-  - name: ens01
+  port: []
 description: Linux bridge with the wrong port
 ipv4:
   address: []
   line 651, in _assert_interfaces_equal\n
current_state.interfaces[ifname],\nlibnmstate.error.NmstateVerificationError:
```

The **NmstateVerificationError** lists the **desired** policy configuration, the **current** configuration of the policy on the node, and the **difference** highlighting the parameters that do not match. In this example, the **port** is included in the **difference**, which suggests that the problem is the port configuration in the policy.

5. To ensure that the policy is configured properly, view the network configuration for one or all of the nodes by requesting the **NodeNetworkState** object. The following command returns the network configuration for the **control-plane-1** node:

```
$ oc get nns control-plane-1 -o yaml
```

The output shows that the interface name on the nodes is **ens1** but the failed policy incorrectly uses **ens01**:

Example output

```
  - ipv4:
# ...
    name: ens1
    state: up
    type: ethernet
```

6. Correct the error by editing the existing policy:

```
$ oc edit nncp ens01-bridge-testfail
```

```
# ...
      port:
        - name: ens1
```

Save the policy to apply the correction.

7. Check the status of the policy to ensure it updated successfully:

```
$ oc get nncp
```

Example output

```
NAME                STATUS
ens01-bridge-testfail   SuccessfullyConfigured
```

The updated policy is successfully configured on all nodes in the cluster.

CHAPTER 32. CONFIGURING THE CLUSTER-WIDE PROXY

Production environments can deny direct access to the internet and instead have an HTTP or HTTPS proxy available. You can configure OpenShift Container Platform to use a proxy by modifying the Proxy object for existing clusters or by configuring the proxy settings in the **install-config.yaml** file for new clusters.

32.1. PREREQUISITES

- Review the sites that your cluster requires access to and determine whether any of them must bypass the proxy. By default, all cluster system egress traffic is proxied, including calls to the cloud provider API for the cloud that hosts your cluster. System-wide proxy affects system components only, not user workloads. Add sites to the Proxy object's **spec.noProxy** field to bypass the proxy if necessary.

> **NOTE**
>
> The Proxy object **status.noProxy** field is populated with the values of the **networking.machineNetwork[].cidr**, **networking.clusterNetwork[].cidr**, and **networking.serviceNetwork[]** fields from your installation configuration with most installation types.
>
> For installations on Amazon Web Services (AWS), Google Cloud Platform (GCP), Microsoft Azure, and Red Hat OpenStack Platform (RHOSP), the **Proxy** object **status.noProxy** field is also populated with the instance metadata endpoint (**169.254.169.254**).
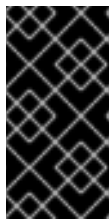
> **IMPORTANT**
>
> If your installation type does not include setting the **networking.machineNetwork[].cidr** field, you must include the machine IP addresses manually in the **.status.noProxy** field to make sure that the traffic between nodes can bypass the proxy.

32.2. ENABLING THE CLUSTER-WIDE PROXY

The **Proxy** object is used to manage the cluster-wide egress proxy. When a cluster is installed or upgraded without the proxy configured, a **Proxy** object is still generated but it will have a nil **spec**. For example:

```
apiVersion: config.openshift.io/v1
kind: Proxy
metadata:
  name: cluster
spec:
  trustedCA:
    name: ""
status:
```

A cluster administrator can configure the proxy for OpenShift Container Platform by modifying this **cluster Proxy** object.

> **NOTE**
>
> Only the **Proxy** object named **cluster** is supported, and no additional proxies can be created.

Prerequisites

- Cluster administrator permissions

- OpenShift Container Platform **oc** CLI tool installed

Procedure

1. Create a config map that contains any additional CA certificates required for proxying HTTPS connections.

   > **NOTE**
   >
   > You can skip this step if the proxy's identity certificate is signed by an authority from the RHCOS trust bundle.

   a. Create a file called **user-ca-bundle.yaml** with the following contents, and provide the values of your PEM-encoded certificates:

   ```
   apiVersion: v1
   data:
     ca-bundle.crt: |        1
       <MY_PEM_ENCODED_CERTS>    2
   kind: ConfigMap
   metadata:
     name: user-ca-bundle    3
     namespace: openshift-config    4
   ```

   1   This data key must be named **ca-bundle.crt**.

   2   One or more PEM-encoded X.509 certificates used to sign the proxy's identity certificate.

   3   The config map name that will be referenced from the **Proxy** object.

   4   The config map must be in the **openshift-config** namespace.

   b. Create the config map from this file:

   ```
   $ oc create -f user-ca-bundle.yaml
   ```

2. Use the **oc edit** command to modify the **Proxy** object:

   ```
   $ oc edit proxy/cluster
   ```

3. Configure the necessary fields for the proxy:

   ```
   apiVersion: config.openshift.io/v1
   ```

```
kind: Proxy
metadata:
  name: cluster
spec:
  httpProxy: http://<username>:<pswd>@<ip>:<port>    1
  httpsProxy: https://<username>:<pswd>@<ip>:<port>  2
  noProxy: example.com  3
  readinessEndpoints:
  - http://www.google.com  4
  - https://www.google.com
  trustedCA:
    name: user-ca-bundle  5
```

1 A proxy URL to use for creating HTTP connections outside the cluster. The URL scheme must be **http**.

2 A proxy URL to use for creating HTTPS connections outside the cluster. The URL scheme must be either **http** or **https**. Specify a URL for the proxy that supports the URL scheme. For example, most proxies will report an error if they are configured to use **https** but they only support **http**. This failure message may not propagate to the logs and can appear to be a network connection failure instead. If using a proxy that listens for **https** connections from the cluster, you may need to configure the cluster to accept the CAs and certificates that the proxy uses.

3 A comma-separated list of destination domain names, domains, IP addresses or other network CIDRs to exclude proxying.

Preface a domain with **.** to match subdomains only. For example, **.y.com** matches **x.y.com**, but not **y.com**. Use ***** to bypass proxy for all destinations. If you scale up workers that are not included in the network defined by the **networking.machineNetwork[].cidr** field from the installation configuration, you must add them to this list to prevent connection issues.

This field is ignored if neither the **httpProxy** or **httpsProxy** fields are set.

4 One or more URLs external to the cluster to use to perform a readiness check before writing the **httpProxy** and **httpsProxy** values to status.

5 A reference to the config map in the **openshift-config** namespace that contains additional CA certificates required for proxying HTTPS connections. Note that the config map must already exist before referencing it here. This field is required unless the proxy's identity certificate is signed by an authority from the RHCOS trust bundle.

4. Save the file to apply the changes.

32.3. REMOVING THE CLUSTER-WIDE PROXY

The **cluster** Proxy object cannot be deleted. To remove the proxy from a cluster, remove all **spec** fields from the Proxy object.

Prerequisites

- Cluster administrator permissions

- OpenShift Container Platform **oc** CLI tool installed

Procedure

1. Use the **oc edit** command to modify the proxy:

   ```
   $ oc edit proxy/cluster
   ```

2. Remove all **spec** fields from the Proxy object. For example:

   ```
   apiVersion: config.openshift.io/v1
   kind: Proxy
   metadata:
     name: cluster
   spec: {}
   ```

3. Save the file to apply the changes.

Additional resources

- [Replacing the CA Bundle certificate](#)

- [Proxy certificate customization](#)

CHAPTER 33. CONFIGURING A CUSTOM PKI

Some platform components, such as the web console, use Routes for communication and must trust other components' certificates to interact with them. If you are using a custom public key infrastructure (PKI), you must configure it so its privately signed CA certificates are recognized across the cluster.

You can leverage the Proxy API to add cluster-wide trusted CA certificates. You must do this either during installation or at runtime.

- During *installation*, configure the cluster-wide proxy. You must define your privately signed CA certificates in the **install-config.yaml** file's **additionalTrustBundle** setting.
  The installation program generates a ConfigMap that is named **user-ca-bundle** that contains the additional CA certificates you defined. The Cluster Network Operator then creates a **trusted-ca-bundle** ConfigMap that merges these CA certificates with the Red Hat Enterprise Linux CoreOS (RHCOS) trust bundle; this ConfigMap is referenced in the Proxy object's **trustedCA** field.

- At *runtime*, modify the default Proxy object to include your privately signed CA certificates (part of cluster's proxy enablement workflow). This involves creating a ConfigMap that contains the privately signed CA certificates that should be trusted by the cluster, and then modifying the proxy resource with the **trustedCA** referencing the privately signed certificates' ConfigMap.

> **NOTE**
>
> The installer configuration's **additionalTrustBundle** field and the proxy resource's **trustedCA** field are used to manage the cluster-wide trust bundle; **additionalTrustBundle** is used at install time and the proxy's **trustedCA** is used at runtime.
>
> The **trustedCA** field is a reference to a **ConfigMap** containing the custom certificate and key pair used by the cluster component.

33.1. CONFIGURING THE CLUSTER-WIDE PROXY DURING INSTALLATION

Production environments can deny direct access to the internet and instead have an HTTP or HTTPS proxy available. You can configure a new OpenShift Container Platform cluster to use a proxy by configuring the proxy settings in the **install-config.yaml** file.

Prerequisites

- You have an existing **install-config.yaml** file.

- You reviewed the sites that your cluster requires access to and determined whether any of them need to bypass the proxy. By default, all cluster egress traffic is proxied, including calls to hosting cloud provider APIs. You added sites to the **Proxy** object's **spec.noProxy** field to bypass the proxy if necessary.

> **NOTE**
>
> The **Proxy** object **status.noProxy** field is populated with the values of the **networking.machineNetwork[].cidr**, **networking.clusterNetwork[].cidr**, and **networking.serviceNetwork[]** fields from your installation configuration.
>
> For installations on Amazon Web Services (AWS), Google Cloud Platform (GCP), Microsoft Azure, and Red Hat OpenStack Platform (RHOSP), the **Proxy** object **status.noProxy** field is also populated with the instance metadata endpoint (**169.254.169.254**).

Procedure

1. Edit your **install-config.yaml** file and add the proxy settings. For example:

```
apiVersion: v1
baseDomain: my.domain.com
proxy:
  httpProxy: http://<username>:<pswd>@<ip>:<port> 1
  httpsProxy: https://<username>:<pswd>@<ip>:<port> 2
  noProxy: ec2.<aws_region>.amazonaws.com,elasticloadbalancing.
<aws_region>.amazonaws.com,s3.<aws_region>.amazonaws.com 3
additionalTrustBundle: | 4
    -----BEGIN CERTIFICATE-----
    <MY_TRUSTED_CA_CERT>
    -----END CERTIFICATE-----
additionalTrustBundlePolicy: <policy_to_add_additionalTrustBundle> 5
```

1 A proxy URL to use for creating HTTP connections outside the cluster. The URL scheme must be **http**.

2 A proxy URL to use for creating HTTPS connections outside the cluster.

3 A comma-separated list of destination domain names, IP addresses, or other network CIDRs to exclude from proxying. Preface a domain with **.** to match subdomains only. For example, **.y.com** matches **x.y.com**, but not **y.com**. Use ***** to bypass the proxy for all destinations. If you have added the Amazon **EC2**,**Elastic Load Balancing**, and **S3** VPC endpoints to your VPC, you must add these endpoints to the **noProxy** field.

4 If provided, the installation program generates a config map that is named **user-ca-bundle** in the **openshift-config** namespace that contains one or more additional CA certificates that are required for proxying HTTPS connections. The Cluster Network Operator then creates a **trusted-ca-bundle** config map that merges these contents with the Red Hat Enterprise Linux CoreOS (RHCOS) trust bundle, and this config map is referenced in the **trustedCA** field of the **Proxy** object. The **additionalTrustBundle** field is required unless the proxy's identity certificate is signed by an authority from the RHCOS trust bundle.

5 Optional: The policy to determine the configuration of the **Proxy** object to reference the **user-ca-bundle** config map in the **trustedCA** field. The allowed values are **Proxyonly** and **Always**. Use **Proxyonly** to reference the **user-ca-bundle** config map only when **http/https** proxy is configured. Use **Always** to always reference the **user-ca-bundle** config map. The default value is **Proxyonly**.

NOTE

The installation program does not support the proxy **readinessEndpoints** field.
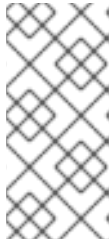
NOTE

If the installer times out, restart and then complete the deployment by using the **wait-for** command of the installer. For example:

```
$ ./openshift-install wait-for install-complete --log-level debug
```

2. Save the file and reference it when installing OpenShift Container Platform.

The installation program creates a cluster-wide proxy that is named **cluster** that uses the proxy settings in the provided **install-config.yaml** file. If no proxy settings are provided, a **cluster Proxy** object is still created, but it will have a nil **spec**.
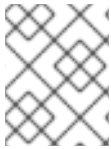
NOTE

Only the **Proxy** object named **cluster** is supported, and no additional proxies can be created.

33.2. ENABLING THE CLUSTER-WIDE PROXY

The **Proxy** object is used to manage the cluster-wide egress proxy. When a cluster is installed or upgraded without the proxy configured, a **Proxy** object is still generated but it will have a nil **spec**. For example:

```
apiVersion: config.openshift.io/v1
kind: Proxy
metadata:
  name: cluster
spec:
  trustedCA:
    name: ""
status:
```

A cluster administrator can configure the proxy for OpenShift Container Platform by modifying this **cluster Proxy** object.

NOTE

Only the **Proxy** object named **cluster** is supported, and no additional proxies can be created.

Prerequisites

- Cluster administrator permissions

- OpenShift Container Platform **oc** CLI tool installed

Procedure

1. Create a config map that contains any additional CA certificates required for proxying HTTPS connections.

> **NOTE**
>
> You can skip this step if the proxy's identity certificate is signed by an authority from the RHCOS trust bundle.

a. Create a file called **user-ca-bundle.yaml** with the following contents, and provide the values of your PEM-encoded certificates:

```
apiVersion: v1
data:
  ca-bundle.crt: | 1
    <MY_PEM_ENCODED_CERTS> 2
kind: ConfigMap
metadata:
  name: user-ca-bundle 3
  namespace: openshift-config 4
```

1 This data key must be named **ca-bundle.crt**.

2 One or more PEM-encoded X.509 certificates used to sign the proxy's identity certificate.

3 The config map name that will be referenced from the **Proxy** object.

4 The config map must be in the **openshift-config** namespace.

b. Create the config map from this file:

```
$ oc create -f user-ca-bundle.yaml
```

2. Use the **oc edit** command to modify the **Proxy** object:

```
$ oc edit proxy/cluster
```

3. Configure the necessary fields for the proxy:

```
apiVersion: config.openshift.io/v1
kind: Proxy
metadata:
  name: cluster
spec:
  httpProxy: http://<username>:<pswd>@<ip>:<port> 1
  httpsProxy: https://<username>:<pswd>@<ip>:<port> 2
  noProxy: example.com 3
  readinessEndpoints:
  - http://www.google.com 4
  - https://www.google.com
  trustedCA:
    name: user-ca-bundle 5
```

1. A proxy URL to use for creating HTTP connections outside the cluster. The URL scheme must be **http**.

2. A proxy URL to use for creating HTTPS connections outside the cluster. The URL scheme must be either **http** or **https**. Specify a URL for the proxy that supports the URL scheme. For example, most proxies will report an error if they are configured to use **https** but they only support **http**. This failure message may not propagate to the logs and can appear to be a network connection failure instead. If using a proxy that listens for **https** connections from the cluster, you may need to configure the cluster to accept the CAs and certificates that the proxy uses.

3. A comma-separated list of destination domain names, domains, IP addresses or other network CIDRs to exclude proxying.

   Preface a domain with **.** to match subdomains only. For example, **.y.com** matches **x.y.com**, but not **y.com**. Use ***** to bypass proxy for all destinations. If you scale up workers that are not included in the network defined by the **networking.machineNetwork[].cidr** field from the installation configuration, you must add them to this list to prevent connection issues.

   This field is ignored if neither the **httpProxy** or **httpsProxy** fields are set.

4. One or more URLs external to the cluster to use to perform a readiness check before writing the **httpProxy** and **httpsProxy** values to status.

5. A reference to the config map in the **openshift-config** namespace that contains additional CA certificates required for proxying HTTPS connections. Note that the config map must already exist before referencing it here. This field is required unless the proxy's identity certificate is signed by an authority from the RHCOS trust bundle.

4. Save the file to apply the changes.

33.3. CERTIFICATE INJECTION USING OPERATORS

Once your custom CA certificate is added to the cluster via ConfigMap, the Cluster Network Operator merges the user-provided and system CA certificates into a single bundle and injects the merged bundle into the Operator requesting the trust bundle injection.

IMPORTANT

After adding a **config.openshift.io/inject-trusted-cabundle="true"** label to the config map, existing data in it is deleted. The Cluster Network Operator takes ownership of a config map and only accepts **ca-bundle** as data. You must use a separate config map to store **service-ca.crt** by using the **service.beta.openshift.io/inject-cabundle=true** annotation or a similar configuration. Adding a **config.openshift.io/inject-trusted-cabundle="true"** label and **service.beta.openshift.io/inject-cabundle=true** annotation on the same config map can cause issues.

Operators request this injection by creating an empty ConfigMap with the following label:

```
config.openshift.io/inject-trusted-cabundle="true"
```

An example of the empty ConfigMap:

```
apiVersion: v1
```

```
data: {}
kind: ConfigMap
metadata:
  labels:
    config.openshift.io/inject-trusted-cabundle: "true"
  name: ca-inject 1
  namespace: apache
```

1   Specifies the empty ConfigMap name.

The Operator mounts this ConfigMap into the container's local trust store.
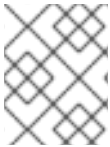
> **NOTE**
>
> Adding a trusted CA certificate is only needed if the certificate is not included in the Red Hat Enterprise Linux CoreOS (RHCOS) trust bundle.

Certificate injection is not limited to Operators. The Cluster Network Operator injects certificates across any namespace when an empty ConfigMap is created with the **config.openshift.io/inject-trusted-cabundle=true** label.

The ConfigMap can reside in any namespace, but the ConfigMap must be mounted as a volume to each container within a pod that requires a custom CA. For example:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-example-custom-ca-deployment
  namespace: my-example-custom-ca-ns
spec:
  ...
    spec:
      ...
      containers:
        - name: my-container-that-needs-custom-ca
          volumeMounts:
          - name: trusted-ca
            mountPath: /etc/pki/ca-trust/extracted/pem
            readOnly: true
      volumes:
      - name: trusted-ca
        configMap:
          name: trusted-ca
          items:
            - key: ca-bundle.crt 1
              path: tls-ca-bundle.pem 2
```

1   **ca-bundle.crt** is required as the ConfigMap key.

2   **tls-ca-bundle.pem** is required as the ConfigMap path.

CHAPTER 34. LOAD BALANCING ON RHOSP

34.1. LIMITATIONS OF LOAD BALANCER SERVICES

OpenShift Container Platform clusters on Red Hat OpenStack Platform (RHOSP) use Octavia to handle load balancer services. As a result of this choice, such clusters have a number of functional limitations.

RHOSP Octavia has two supported providers: Amphora and OVN. These providers differ in terms of available features as well as implementation details. These distinctions affect load balancer services that are created on your cluster.

34.1.1. Local external traffic policies

You can set the external traffic policy (ETP) parameter, **.spec.externalTrafficPolicy**, on a load balancer service to preserve the source IP address of incoming traffic when it reaches service endpoint pods. However, if your cluster uses the Amphora Octavia provider, the source IP of the traffic is replaced with the IP address of the Amphora VM. This behavior does not occur if your cluster uses the OVN Octavia provider.

Having the **ETP** option set to **Local** requires that health monitors be created for the load balancer. Without health monitors, traffic can be routed to a node that doesn't have a functional endpoint, which causes the connection to drop. To force Cloud Provider OpenStack to create health monitors, you must set the value of the **create-monitor** option in the cloud provider configuration to **true**.

In RHOSP 16.2, the OVN Octavia provider does not support health monitors. Therefore, setting the ETP to local is unsupported.

In RHOSP 16.2, the Amphora Octavia provider does not support HTTP monitors on UDP pools. As a result, UDP load balancer services have **UDP-CONNECT** monitors created instead. Due to implementation details, this configuration only functions properly with the OVN-Kubernetes CNI plugin. When the OpenShift SDN CNI plugin is used, the UDP services alive nodes are detected unreliably.

34.1.2. Load balancer source ranges

Use the **.spec.loadBalancerSourceRanges** property to restrict the traffic that can pass through the load balancer according to source IP. This property is supported for use with the Amphora Octavia provider only. If your cluster uses the OVN Octavia provider, the option is ignored and traffic is unrestricted.

34.2. USING THE OCTAVIA OVN LOAD BALANCER PROVIDER DRIVER WITH KURYR SDN

> **IMPORTANT**
>
> Kuryr is a deprecated feature. Deprecated functionality is still included in OpenShift Container Platform and continues to be supported; however, it will be removed in a future release of this product and is not recommended for new deployments.
>
> For the most recent list of major functionality that has been deprecated or removed within OpenShift Container Platform, refer to the *Deprecated and removed features* section of the OpenShift Container Platform release notes.

If your OpenShift Container Platform cluster uses Kuryr and was installed on a Red Hat OpenStack Platform (RHOSP) 13 cloud that was later upgraded to RHOSP 16, you can configure it to use the Octavia OVN provider driver.

> **IMPORTANT**
>
> Kuryr replaces existing load balancers after you change provider drivers. This process results in some downtime.

Prerequisites

- Install the RHOSP CLI, **openstack**.

- Install the OpenShift Container Platform CLI, **oc**.

- Verify that the Octavia OVN driver on RHOSP is enabled.

  > **TIP**
  >
  > To view a list of available Octavia drivers, on a command line, enter **openstack loadbalancer provider list**.
  >
  > The **ovn** driver is displayed in the command's output.

Procedure

To change from the Octavia Amphora provider driver to Octavia OVN:

1. Open the **kuryr-config** ConfigMap. On a command line, enter:

   ```
   $ oc -n openshift-kuryr edit cm kuryr-config
   ```

2. In the ConfigMap, delete the line that contains **kuryr-octavia-provider: default**. For example:

   ```
   ...
   kind: ConfigMap
   metadata:
     annotations:
       networkoperator.openshift.io/kuryr-octavia-provider: default    1
   ...
   ```

   1 Delete this line. The cluster will regenerate it with **ovn** as the value.

   Wait for the Cluster Network Operator to detect the modification and to redeploy the **kuryr-controller** and **kuryr-cni** pods. This process might take several minutes.

3. Verify that the **kuryr-config** ConfigMap annotation is present with **ovn** as its value. On a command line, enter:

   ```
   $ oc -n openshift-kuryr edit cm kuryr-config
   ```

   The **ovn** provider value is displayed in the output:

   ```
   ...
   ```

```
kind: ConfigMap
metadata:
  annotations:
    networkoperator.openshift.io/kuryr-octavia-provider: ovn
...
```

4. Verify that RHOSP recreated its load balancers.

   a. On a command line, enter:

      ```
      $ openstack loadbalancer list | grep amphora
      ```

      A single Amphora load balancer is displayed. For example:

      ```
      a4db683b-2b7b-4988-a582-c39daaad7981 | ostest-7mbj6-kuryr-api-loadbalancer  |
      84c99c906edd475ba19478a9a6690efd | 172.30.0.1     | ACTIVE           | amphora
      ```

   b. Search for **ovn** load balancers by entering:

      ```
      $ openstack loadbalancer list | grep ovn
      ```

      The remaining load balancers of the **ovn** type are displayed. For example:

      ```
      2dffe783-98ae-4048-98d0-32aa684664cc | openshift-apiserver-operator/metrics |
      84c99c906edd475ba19478a9a6690efd | 172.30.167.119 | ACTIVE           | ovn
      0b1b2193-251f-4243-af39-2f99b29d18c5 | openshift-etcd/etcd             |
      84c99c906edd475ba19478a9a6690efd | 172.30.143.226 | ACTIVE           | ovn
      f05b07fc-01b7-4673-bd4d-adaa4391458e | openshift-dns-operator/metrics     |
      84c99c906edd475ba19478a9a6690efd | 172.30.152.27  | ACTIVE           | ovn
      ```

34.3. SCALING CLUSTERS FOR APPLICATION TRAFFIC BY USING OCTAVIA

OpenShift Container Platform clusters that run on Red Hat OpenStack Platform (RHOSP) can use the Octavia load balancing service to distribute traffic across multiple virtual machines (VMs) or floating IP addresses. This feature mitigates the bottleneck that single machines or addresses create.

If your cluster uses Kuryr, the Cluster Network Operator created an internal Octavia load balancer at deployment. You can use this load balancer for application network scaling.

If your cluster does not use Kuryr, you must create your own Octavia load balancer to use it for application network scaling.

34.3.1. Scaling clusters by using Octavia

If you want to use multiple API load balancers, or if your cluster does not use Kuryr, create an Octavia load balancer and then configure your cluster to use it.

Prerequisites

- Octavia is available on your Red Hat OpenStack Platform (RHOSP) deployment.

Procedure

1. From a command line, create an Octavia load balancer that uses the Amphora driver:

   ```
   $ openstack loadbalancer create --name API_OCP_CLUSTER --vip-subnet-id
   <id_of_worker_vms_subnet>
   ```

   You can use a name of your choice instead of **API_OCP_CLUSTER**.

2. After the load balancer becomes active, create listeners:

   ```
   $ openstack loadbalancer listener create --name API_OCP_CLUSTER_6443 --protocol
   HTTPS--protocol-port 6443 API_OCP_CLUSTER
   ```
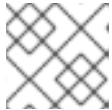
   > **NOTE**
   >
   > To view the status of the load balancer, enter **openstack loadbalancer list**.

3. Create a pool that uses the round robin algorithm and has session persistence enabled:

   ```
   $ openstack loadbalancer pool create --name API_OCP_CLUSTER_pool_6443 --lb-
   algorithm ROUND_ROBIN --session-persistence type=<source_IP_address> --listener
   API_OCP_CLUSTER_6443 --protocol HTTPS
   ```

4. To ensure that control plane machines are available, create a health monitor:

   ```
   $ openstack loadbalancer healthmonitor create --delay 5 --max-retries 4 --timeout 10 --type
   TCP API_OCP_CLUSTER_pool_6443
   ```

5. Add the control plane machines as members of the load balancer pool:

   ```
   $ for SERVER in $(MASTER-0-IP MASTER-1-IP MASTER-2-IP)
   do
     openstack loadbalancer member create --address $SERVER  --protocol-port 6443
   API_OCP_CLUSTER_pool_6443
   done
   ```

6. Optional: To reuse the cluster API floating IP address, unset it:

   ```
   $ openstack floating ip unset $API_FIP
   ```

7. Add either the unset **API_FIP** or a new address to the created load balancer VIP:

   ```
   $ openstack floating ip set  --port $(openstack loadbalancer show -c <vip_port_id> -f value
   API_OCP_CLUSTER) $API_FIP
   ```

Your cluster now uses Octavia for load balancing.

> **NOTE**
>
> If Kuryr uses the Octavia Amphora driver, all traffic is routed through a single Amphora virtual machine (VM).
>
> You can repeat this procedure to create additional load balancers, which can alleviate the bottleneck.

34.3.2. Scaling clusters that use Kuryr by using Octavia

> **IMPORTANT**
>
> Kuryr is a deprecated feature. Deprecated functionality is still included in OpenShift Container Platform and continues to be supported; however, it will be removed in a future release of this product and is not recommended for new deployments.
>
> For the most recent list of major functionality that has been deprecated or removed within OpenShift Container Platform, refer to the *Deprecated and removed features* section of the OpenShift Container Platform release notes.

If your cluster uses Kuryr, associate the API floating IP address of your cluster with the pre-existing Octavia load balancer.

Prerequisites

- Your OpenShift Container Platform cluster uses Kuryr.

- Octavia is available on your Red Hat OpenStack Platform (RHOSP) deployment.

Procedure

1. Optional: From a command line, to reuse the cluster API floating IP address, unset it:

   ```
   $ openstack floating ip unset $API_FIP
   ```

2. Add either the unset **API_FIP** or a new address to the created load balancer VIP:

   ```
   $ openstack floating ip set --port $(openstack loadbalancer show -c <vip_port_id> -f value ${OCP_CLUSTER}-kuryr-api-loadbalancer) $API_FIP
   ```

Your cluster now uses Octavia for load balancing.

> **NOTE**
>
> If Kuryr uses the Octavia Amphora driver, all traffic is routed through a single Amphora virtual machine (VM).
>
> You can repeat this procedure to create additional load balancers, which can alleviate the bottleneck.

34.4. SCALING FOR INGRESS TRAFFIC BY USING RHOSP OCTAVIA

> **IMPORTANT**
>
> Kuryr is a deprecated feature. Deprecated functionality is still included in OpenShift Container Platform and continues to be supported; however, it will be removed in a future release of this product and is not recommended for new deployments.
>
> For the most recent list of major functionality that has been deprecated or removed within OpenShift Container Platform, refer to the *Deprecated and removed features* section of the OpenShift Container Platform release notes.

You can use Octavia load balancers to scale Ingress controllers on clusters that use Kuryr.

Prerequisites

- Your OpenShift Container Platform cluster uses Kuryr.

- Octavia is available on your RHOSP deployment.

Procedure

1. To copy the current internal router service, on a command line, enter:

   ```
   $ oc -n openshift-ingress get svc router-internal-default -o yaml > external_router.yaml
   ```

2. In the file **external_router.yaml**, change the values of **metadata.name** and **spec.type** to **LoadBalancer**.

   Example router file

   ```
   apiVersion: v1
   kind: Service
   metadata:
     labels:
       ingresscontroller.operator.openshift.io/owning-ingresscontroller: default
     name: router-external-default 1
     namespace: openshift-ingress
   spec:
     ports:
     - name: http
       port: 80
       protocol: TCP
       targetPort: http
     - name: https
       port: 443
       protocol: TCP
       targetPort: https
     - name: metrics
       port: 1936
       protocol: TCP
       targetPort: 1936
     selector:
       ingresscontroller.operator.openshift.io/deployment-ingresscontroller: default
     sessionAffinity: None
     type: LoadBalancer 2
   ```

1 Ensure that this value is descriptive, like **router-external-default**.

2 Ensure that this value is **LoadBalancer**.

> **NOTE**
>
> You can delete timestamps and other information that is irrelevant to load balancing.

1. From a command line, create a service from the **external_router.yaml** file:

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

2. Verify that the external IP address of the service is the same as the one that is associated with the load balancer:

   a. On a command line, retrieve the external IP address of the service:

   ```
   $ oc -n openshift-ingress get svc
   ```

   Example output

   ```
   NAME                    TYPE         CLUSTER-IP      EXTERNAL-IP   PORT(S)
   AGE
   router-external-default   LoadBalancer   172.30.235.33    10.46.22.161
   80:30112/TCP,443:32359/TCP,1936:30317/TCP   3m38s
   router-internal-default   ClusterIP      172.30.115.123   <none>
   80/TCP,443/TCP,1936/TCP                 22h
   ```

   b. Retrieve the IP address of the load balancer:

   ```
   $ openstack loadbalancer list | grep router-external
   ```

   Example output

   ```
   | 21bf6afe-b498-4a16-a958-3229e83c002c | openshift-ingress/router-external-default |
   66f3816acf1b431691b8d132cc9d793c | 172.30.235.33  | ACTIVE | octavia |
   ```

   c. Verify that the addresses you retrieved in the previous steps are associated with each other in the floating IP list:

   ```
   $ openstack floating ip list | grep 172.30.235.33
   ```

   Example output

   ```
   | e2f80e97-8266-4b69-8636-e58bacf1879e | 10.46.22.161 | 172.30.235.33 | 655e7122-
   806a-4e0a-a104-220c6e17bda6 | a565e55a-99e7-4d15-b4df-f9d7ee8c9deb |
   66f3816acf1b431691b8d132cc9d793c |
   ```

You can now use the value of **EXTERNAL-IP** as the new Ingress address.
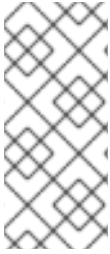
NOTE

If Kuryr uses the Octavia Amphora driver, all traffic is routed through a single Amphora virtual machine (VM).

You can repeat this procedure to create additional load balancers, which can alleviate the bottleneck.

34.5. SERVICES FOR AN EXTERNAL LOAD BALANCER

You can configure an OpenShift Container Platform cluster on Red Hat OpenStack Platform (RHOSP) to use an external load balancer in place of the default load balancer.

IMPORTANT

Configuring an external load balancer depends on your vendor's load balancer.

The information and examples in this section are for guideline purposes only. Consult the vendor documentation for more specific information about the vendor's load balancer.

Red Hat supports the following services for an external load balancer:

- Ingress Controller

- OpenShift API

- OpenShift MachineConfig API

You can choose whether you want to configure one or all of these services for an external load balancer. Configuring only the Ingress Controller service is a common configuration option. To better understand each service, view the following diagrams:

Figure 34.1. Example network workflow that shows an Ingress Controller operating in an OpenShift Container Platform environment
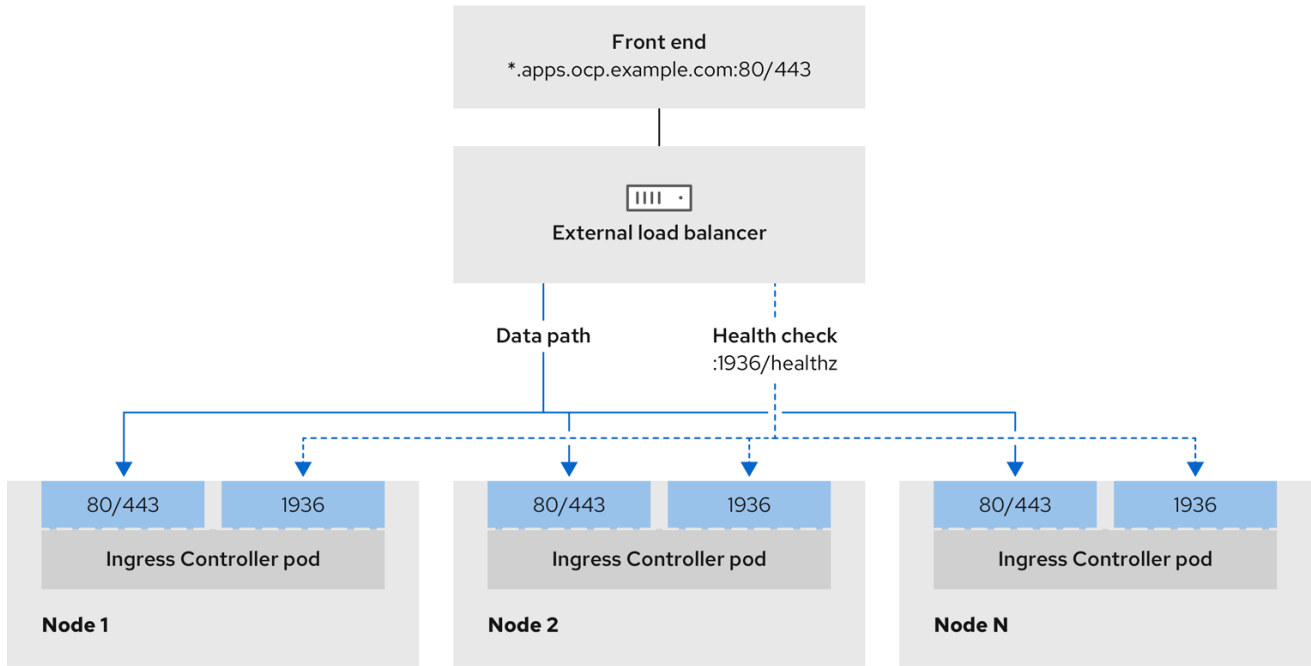
Figure 34.2. Example network workflow that shows an OpenShift API operating in an OpenShift Container Platform environment
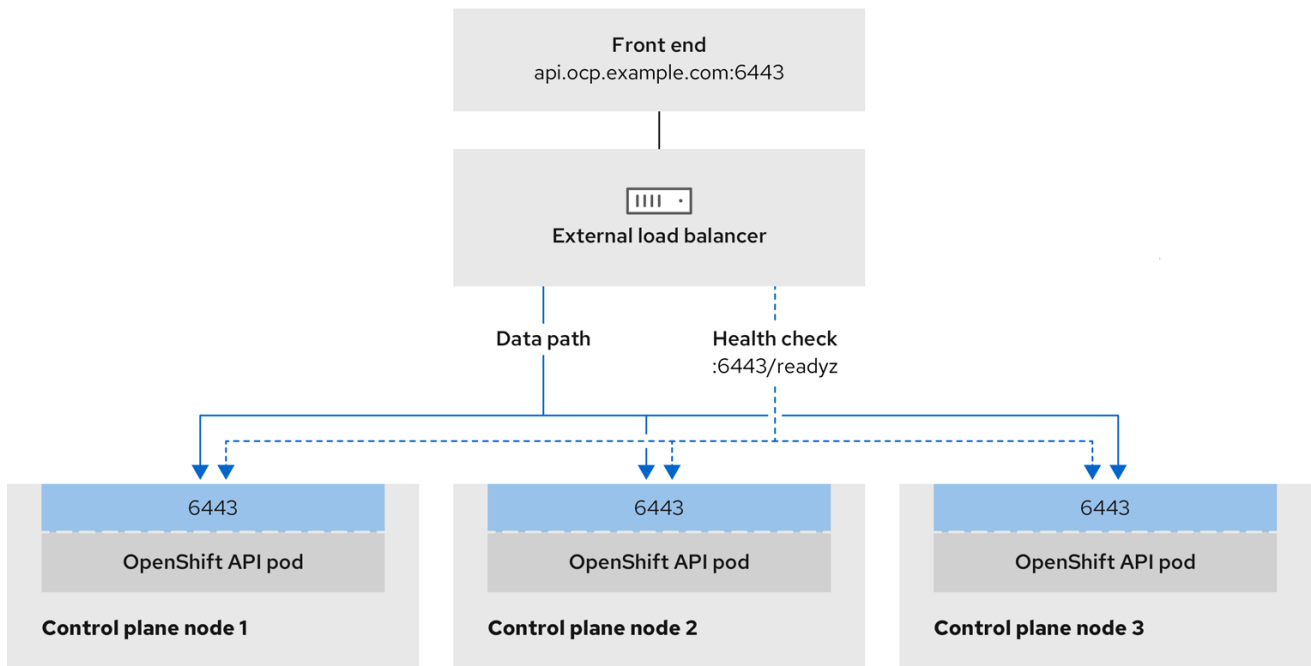
Figure 34.3. Example network workflow that shows an OpenShift MachineConfig API operating in an OpenShift Container Platform environment
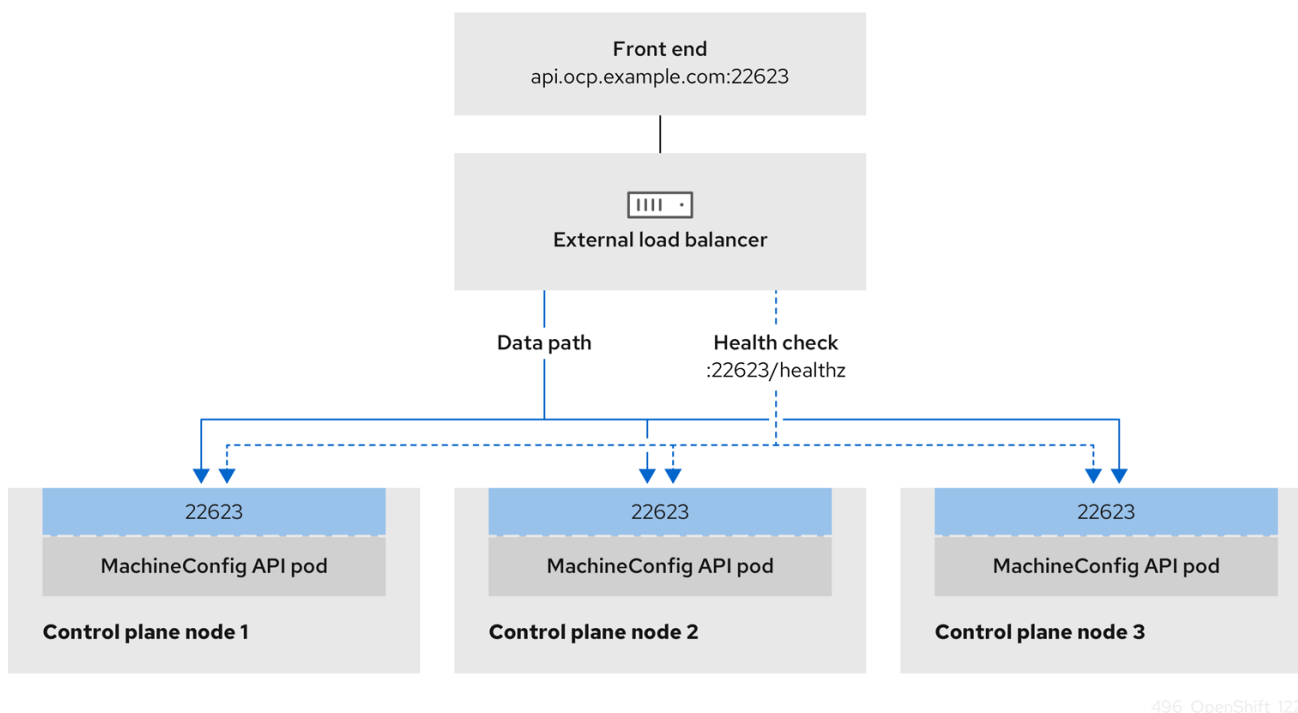
The following configuration options are supported for external load balancers:

- Use a node selector to map the Ingress Controller to a specific set of nodes. You must assign a static IP address to each node in this set, or configure each node to receive the same IP address from the Dynamic Host Configuration Protocol (DHCP). Infrastructure nodes commonly receive this type of configuration.

- Target all IP addresses on a subnet. This configuration can reduce maintenance overhead, because you can create and destroy nodes within those networks without reconfiguring the load balancer targets. If you deploy your ingress pods by using a machine set on a smaller network, such as a **/27** or **/28**, you can simplify your load balancer targets.

TIP

You can list all IP addresses that exist in a network by checking the machine config pool's resources.

Before you configure an external load balancer for your OpenShift Container Platform cluster, consider the following information:

- For a front-end IP address, you can use the same IP address for the front-end IP address, the Ingress Controller's load balancer, and API load balancer. Check the vendor's documentation for this capability.

- For a back-end IP address, ensure that an IP address for an OpenShift Container Platform control plane node does not change during the lifetime of the external load balancer. You can achieve this by completing one of the following actions:

  - Assign a static IP address to each control plane node.

○ Configure each node to receive the same IP address from the DHCP every time the node requests a DHCP lease. Depending on the vendor, the DHCP lease might be in the form of an IP reservation or a static DHCP assignment.

- Manually define each node that runs the Ingress Controller in the external load balancer for the Ingress Controller back-end service. For example, if the Ingress Controller moves to an undefined node, a connection outage can occur.

34.5.1. Configuring an external load balancer

You can configure an OpenShift Container Platform cluster on Red Hat OpenStack Platform (RHOSP) to use an external load balancer in place of the default load balancer.
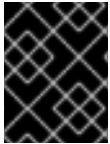
> **IMPORTANT**
>
> Before you configure an external load balancer, ensure that you read the "Services for an external load balancer" section.

Read the following prerequisites that apply to the service that you want to configure for your external load balancer.

> **NOTE**
>
> MetalLB, that runs on a cluster, functions as an external load balancer.

OpenShift API prerequisites

- You defined a front-end IP address.

- TCP ports 6443 and 22623 are exposed on the front-end IP address of your load balancer. Check the following items:

  ○ Port 6443 provides access to the OpenShift API service.

  ○ Port 22623 can provide ignition startup configurations to nodes.

- The front-end IP address and port 6443 are reachable by all users of your system with a location external to your OpenShift Container Platform cluster.

- The front-end IP address and port 22623 are reachable only by OpenShift Container Platform nodes.

- The load balancer backend can communicate with OpenShift Container Platform control plane nodes on port 6443 and 22623.

Ingress Controller prerequisites

- You defined a front-end IP address.

- TCP ports 443 and 80 are exposed on the front-end IP address of your load balancer.

- The front-end IP address, port 80 and port 443 are be reachable by all users of your system with a location external to your OpenShift Container Platform cluster.

- The front-end IP address, port 80 and port 443 are reachable to all nodes that operate in your OpenShift Container Platform cluster.

- The load balancer backend can communicate with OpenShift Container Platform nodes that run the Ingress Controller on ports 80, 443, and 1936.

Prerequisite for health check URL specifications

You can configure most load balancers by setting health check URLs that determine if a service is available or unavailable. OpenShift Container Platform provides these health checks for the OpenShift API, Machine Configuration API, and Ingress Controller backend services.

The following examples demonstrate health check specifications for the previously listed backend services:

Example of a Kubernetes API health check specification

```
Path: HTTPS:6443/readyz
Healthy threshold: 2
Unhealthy threshold: 2
Timeout: 10
Interval: 10
```

Example of a Machine Config API health check specification

```
Path: HTTPS:22623/healthz
Healthy threshold: 2
Unhealthy threshold: 2
Timeout: 10
Interval: 10
```

Example of an Ingress Controller health check specification

```
Path: HTTP:1936/healthz/ready
Healthy threshold: 2
Unhealthy threshold: 2
Timeout: 5
Interval: 10
```

Procedure

1. Configure the HAProxy Ingress Controller, so that you can enable access to the cluster from your load balancer on ports 6443, 443, and 80:

   ### Example HAProxy configuration

   ```
   #...
   listen my-cluster-api-6443
       bind 192.168.1.100:6443
       mode tcp
       balance roundrobin
     option httpchk
     http-check connect
     http-check send meth GET uri /readyz
   ```

```
    http-check expect status 200
      server my-cluster-master-2 192.168.1.101:6443 check inter 10s rise 2 fall 2
      server my-cluster-master-0 192.168.1.102:6443 check inter 10s rise 2 fall 2
      server my-cluster-master-1 192.168.1.103:6443 check inter 10s rise 2 fall 2

  listen my-cluster-machine-config-api-22623
      bind 192.168.1.100:22623
      mode tcp
      balance roundrobin
    option httpchk
    http-check connect
    http-check send meth GET uri /healthz
    http-check expect status 200
      server my-cluster-master-2 192.168.1.101:22623 check inter 10s rise 2 fall 2
      server my-cluster-master-0 192.168.1.102:22623 check inter 10s rise 2 fall 2
      server my-cluster-master-1 192.168.1.103:22623 check inter 10s rise 2 fall 2

  listen my-cluster-apps-443
        bind 192.168.1.100:443
        mode tcp
        balance roundrobin
      option httpchk
      http-check connect
      http-check send meth GET uri /healthz/ready
      http-check expect status 200
        server my-cluster-worker-0 192.168.1.111:443 check port 1936 inter 10s rise 2 fall 2
        server my-cluster-worker-1 192.168.1.112:443 check port 1936 inter 10s rise 2 fall 2
        server my-cluster-worker-2 192.168.1.113:443 check port 1936 inter 10s rise 2 fall 2

  listen my-cluster-apps-80
        bind 192.168.1.100:80
        mode tcp
        balance roundrobin
      option httpchk
      http-check connect
      http-check send meth GET uri /healthz/ready
      http-check expect status 200
        server my-cluster-worker-0 192.168.1.111:80 check port 1936 inter 10s rise 2 fall 2
        server my-cluster-worker-1 192.168.1.112:80 check port 1936 inter 10s rise 2 fall 2
        server my-cluster-worker-2 192.168.1.113:80 check port 1936 inter 10s rise 2 fall 2
  # ...
```

2. Use the **curl** CLI command to verify that the external load balancer and its resources are operational:

   a. Verify that the cluster machine configuration API is accessible to the Kubernetes API server resource, by running the following command and observing the response:

   ```
   $ curl https://<loadbalancer_ip_address>:6443/version --insecure
   ```

   If the configuration is correct, you receive a JSON object in response:

   ```
   {
     "major": "1",
     "minor": "11+",
     "gitVersion": "v1.11.0+ad103ed",
   ```

```
  "gitCommit": "ad103ed",
  "gitTreeState": "clean",
  "buildDate": "2019-01-09T06:44:10Z",
  "goVersion": "go1.10.3",
  "compiler": "gc",
  "platform": "linux/amd64"
}
```

b. Verify that the cluster machine configuration API is accessible to the Machine config server resource, by running the following command and observing the output:

```
$ curl -v https://<loadbalancer_ip_address>:22623/healthz --insecure
```

If the configuration is correct, the output from the command shows the following response:

```
HTTP/1.1 200 OK
Content-Length: 0
```

c. Verify that the controller is accessible to the Ingress Controller resource on port 80, by running the following command and observing the output:

```
$ curl -I -L -H "Host: console-openshift-console.apps.<cluster_name>.<base_domain>"
http://<load_balancer_front_end_IP_address>
```

If the configuration is correct, the output from the command shows the following response:

```
HTTP/1.1 302 Found
content-length: 0
location: https://console-openshift-console.apps.ocp4.private.opequon.net/
cache-control: no-cache
```

d. Verify that the controller is accessible to the Ingress Controller resource on port 443, by running the following command and observing the output:

```
$ curl -I -L --insecure --resolve console-openshift-console.apps.<cluster_name>.
<base_domain>:443:<Load Balancer Front End IP Address> https://console-openshift-
console.apps.<cluster_name>.<base_domain>
```

If the configuration is correct, the output from the command shows the following response:

```
HTTP/1.1 200 OK
referrer-policy: strict-origin-when-cross-origin
set-cookie: csrf-
token=UlYWOyQ62LWjw2h003xtYSKlh1a0Py2hhctw0WmV2YEdhJjFyQwWcGBsja261dG
LgaYO0nxzVErhiXt6QepA7g==; Path=/; Secure; SameSite=Lax
x-content-type-options: nosniff
x-dns-prefetch-control: off
x-frame-options: DENY
x-xss-protection: 1; mode=block
date: Wed, 04 Oct 2023 16:29:38 GMT
content-type: text/html; charset=utf-8
set-cookie:
```

```
1e2670d92730b515ce3a1bb65da45062=1bf5e9573c9a2760c964ed1659cc1673; path=/;
HttpOnly; Secure; SameSite=None
cache-control: private
```

3. Configure the DNS records for your cluster to target the front-end IP addresses of the external load balancer. You must update records to your DNS server for the cluster API and applications over the load balancer.

Examples of modified DNS records

```
<load_balancer_ip_address>  A  api.<cluster_name>.<base_domain>
A record pointing to Load Balancer Front End
```

```
<load_balancer_ip_address>   A apps.<cluster_name>.<base_domain>
A record pointing to Load Balancer Front End
```

> **IMPORTANT**
>
> DNS propagation might take some time for each DNS record to become available. Ensure that each DNS record propagates before validating each record.

4. Use the **curl** CLI command to verify that the external load balancer and DNS record configuration are operational:

   a. Verify that you can access the cluster API, by running the following command and observing the output:

   ```
   $ curl https://api.<cluster_name>.<base_domain>:6443/version --insecure
   ```

   If the configuration is correct, you receive a JSON object in response:

   ```
   {
     "major": "1",
     "minor": "11+",
     "gitVersion": "v1.11.0+ad103ed",
     "gitCommit": "ad103ed",
     "gitTreeState": "clean",
     "buildDate": "2019-01-09T06:44:10Z",
     "goVersion": "go1.10.3",
     "compiler": "gc",
     "platform": "linux/amd64"
   }
   ```

   b. Verify that you can access the cluster machine configuration, by running the following command and observing the output:

   ```
   $ curl -v https://api.<cluster_name>.<base_domain>:22623/healthz --insecure
   ```

   If the configuration is correct, the output from the command shows the following response:

   ```
   HTTP/1.1 200 OK
   Content-Length: 0
   ```

c. Verify that you can access each cluster application on port, by running the following command and observing the output:

```
$ curl http://console-openshift-console.apps.<cluster_name>.<base_domain> -I -L --insecure
```

If the configuration is correct, the output from the command shows the following response:

```
HTTP/1.1 302 Found
content-length: 0
location: https://console-openshift-console.apps.<cluster-name>.<base domain>/
cache-control: no-cacheHTTP/1.1 200 OK
referrer-policy: strict-origin-when-cross-origin
set-cookie: csrf-
token=39HoZgztDnzjJkq/JuLJMeoKNXlfiVv2YgZc09c3TBOBU4NI6kDXaJH1LdicNhN1UsQ
Wzon4Dor9GWGfopaTEQ==; Path=/; Secure
x-content-type-options: nosniff
x-dns-prefetch-control: off
x-frame-options: DENY
x-xss-protection: 1; mode=block
date: Tue, 17 Nov 2020 08:42:10 GMT
content-type: text/html; charset=utf-8
set-cookie:
1e2670d92730b515ce3a1bb65da45062=9b714eb87e93cf34853e87a92d6894be; path=/;
HttpOnly; Secure; SameSite=None
cache-control: private
```

d. Verify that you can access each cluster application on port 443, by running the following command and observing the output:

```
$ curl https://console-openshift-console.apps.<cluster_name>.<base_domain> -I -L --insecure
```

If the configuration is correct, the output from the command shows the following response:

```
HTTP/1.1 200 OK
referrer-policy: strict-origin-when-cross-origin
set-cookie: csrf-
token=UlYWOyQ62LWjw2h003xtYSKlh1a0Py2hhctw0WmV2YEdhJjFyQwWcGBsja261dG
LgaYO0nxzVErhiXt6QepA7g==; Path=/; Secure; SameSite=Lax
x-content-type-options: nosniff
x-dns-prefetch-control: off
x-frame-options: DENY
x-xss-protection: 1; mode=block
date: Wed, 04 Oct 2023 16:29:38 GMT
content-type: text/html; charset=utf-8
set-cookie:
1e2670d92730b515ce3a1bb65da45062=1bf5e9573c9a2760c964ed1659cc1673; path=/;
HttpOnly; Secure; SameSite=None
cache-control: private
```

CHAPTER 35. LOAD BALANCING WITH METALLB

35.1. ABOUT METALLB AND THE METALLB OPERATOR

As a cluster administrator, you can add the MetalLB Operator to your cluster so that when a service of type **LoadBalancer** is added to the cluster, MetalLB can add an external IP address for the service. The external IP address is added to the host network for your cluster.

35.1.1. When to use MetalLB

Using MetalLB is valuable when you have a bare-metal cluster, or an infrastructure that is like bare metal, and you want fault-tolerant access to an application through an external IP address.

You must configure your networking infrastructure to ensure that network traffic for the external IP address is routed from clients to the host network for the cluster.

After deploying MetalLB with the MetalLB Operator, when you add a service of type **LoadBalancer**, MetalLB provides a platform-native load balancer.

MetalLB operating in layer2 mode provides support for failover by utilizing a mechanism similar to IP failover. However, instead of relying on the virtual router redundancy protocol (VRRP) and keepalived, MetalLB leverages a gossip-based protocol to identify instances of node failure. When a failover is detected, another node assumes the role of the leader node, and a gratuitous ARP message is dispatched to broadcast this change.

MetalLB operating in layer3 or border gateway protocol (BGP) mode delegates failure detection to the network. The BGP router or routers that the OpenShift Container Platform nodes have established a connection with will identify any node failure and terminate the routes to that node.

Using MetalLB instead of IP failover is preferable for ensuring high availability of pods and services.

35.1.2. MetalLB Operator custom resources

The MetalLB Operator monitors its own namespace for the following custom resources:

MetalLB

> When you add a **MetalLB** custom resource to the cluster, the MetalLB Operator deploys MetalLB on the cluster. The Operator only supports a single instance of the custom resource. If the instance is deleted, the Operator removes MetalLB from the cluster.

IPAddressPool

> MetalLB requires one or more pools of IP addresses that it can assign to a service when you add a service of type **LoadBalancer**. An **IPAddressPool** includes a list of IP addresses. The list can be a single IP address that is set using a range, such as 1.1.1.1-1.1.1.1, a range specified in CIDR notation, a range specified as a starting and ending address separated by a hyphen, or a combination of the three. An **IPAddressPool** requires a name. The documentation uses names like **doc-example**, **doc-example-reserved**, and **doc-example-ipv6**. The MetalLB **controller** assigns IP addresses from a pool of addresses in an **IPAddressPool**. **L2Advertisement** and **BGPAdvertisement** custom resources enable the advertisement of a given IP from a given pool. You can assign IP addresses from an **IPAddressPool** to services and namespaces by using the **spec.serviceAllocation** specification in the **IPAddressPool** custom resource.
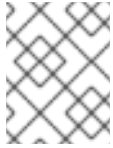
> **NOTE**
>
> A single **IPAddressPool** can be referenced by a L2 advertisement and a BGP advertisement.

BGPPeer

The BGP peer custom resource identifies the BGP router for MetalLB to communicate with, the AS number of the router, the AS number for MetalLB, and customizations for route advertisement. MetalLB advertises the routes for service load-balancer IP addresses to one or more BGP peers.

BFDProfile

The BFD profile custom resource configures Bidirectional Forwarding Detection (BFD) for a BGP peer. BFD provides faster path failure detection than BGP alone provides.

L2Advertisement

The L2Advertisement custom resource advertises an IP coming from an **IPAddressPool** using the L2 protocol.

BGPAdvertisement

The BGPAdvertisement custom resource advertises an IP coming from an **IPAddressPool** using the BGP protocol.

After you add the **MetalLB** custom resource to the cluster and the Operator deploys MetalLB, the **controller** and **speaker** MetalLB software components begin running.

MetalLB validates all relevant custom resources.

35.1.3. MetalLB software components

When you install the MetalLB Operator, the **metallb-operator-controller-manager** deployment starts a pod. The pod is the implementation of the Operator. The pod monitors for changes to all the relevant resources.

When the Operator starts an instance of MetalLB, it starts a **controller** deployment and a **speaker** daemon set.

> **NOTE**
>
> You can configure deployment specifications in the MetalLB custom resource to manage how **controller** and **speaker** pods deploy and run in your cluster. For more information about these deployment specifications, see the *Additional Resources* section.

controller

The Operator starts the deployment and a single pod. When you add a service of type **LoadBalancer**, Kubernetes uses the **controller** to allocate an IP address from an address pool. In case of a service failure, verify you have the following entry in your **controller** pod logs:

Example output

```
"event":"ipAllocated","ip":"172.22.0.201","msg":"IP address assigned by controller
```

speaker

The Operator starts a daemon set for **speaker** pods. By default, a pod is started on each node in

your cluster. You can limit the pods to specific nodes by specifying a node selector in the **MetalLB** custom resource when you start MetalLB. If the **controller** allocated the IP address to the service and service is still unavailable, read the **speaker** pod logs. If the **speaker** pod is unavailable, run the **oc describe pod -n** command.

For layer 2 mode, after the **controller** allocates an IP address for the service, the **speaker** pods use an algorithm to determine which **speaker** pod on which node will announce the load balancer IP address. The algorithm involves hashing the node name and the load balancer IP address. For more information, see "MetalLB and external traffic policy". The **speaker** uses Address Resolution Protocol (ARP) to announce IPv4 addresses and Neighbor Discovery Protocol (NDP) to announce IPv6 addresses.

For Border Gateway Protocol (BGP) mode, after the **controller** allocates an IP address for the service, each **speaker** pod advertises the load balancer IP address with its BGP peers. You can configure which nodes start BGP sessions with BGP peers.

Requests for the load balancer IP address are routed to the node with the **speaker** that announces the IP address. After the node receives the packets, the service proxy routes the packets to an endpoint for the service. The endpoint can be on the same node in the optimal case, or it can be on another node. The service proxy chooses an endpoint each time a connection is established.

35.1.4. MetalLB and external traffic policy

With layer 2 mode, one node in your cluster receives all the traffic for the service IP address. With BGP mode, a router on the host network opens a connection to one of the nodes in the cluster for a new client connection. How your cluster handles the traffic after it enters the node is affected by the external traffic policy.

cluster

> This is the default value for **spec.externalTrafficPolicy**.
>
> With the **cluster** traffic policy, after the node receives the traffic, the service proxy distributes the traffic to all the pods in your service. This policy provides uniform traffic distribution across the pods, but it obscures the client IP address and it can appear to the application in your pods that the traffic originates from the node rather than the client.

local

> With the **local** traffic policy, after the node receives the traffic, the service proxy only sends traffic to the pods on the same node. For example, if the **speaker** pod on node A announces the external service IP, then all traffic is sent to node A. After the traffic enters node A, the service proxy only sends traffic to pods for the service that are also on node A. Pods for the service that are on additional nodes do not receive any traffic from node A. Pods for the service on additional nodes act as replicas in case failover is needed.
>
> This policy does not affect the client IP address. Application pods can determine the client IP address from the incoming connections.

NOTE

The following information is important when configuring the external traffic policy in BGP mode.

Although MetalLB advertises the load balancer IP address from all the eligible nodes, the number of nodes loadbalancing the service can be limited by the capacity of the router to establish equal-cost multipath (ECMP) routes. If the number of nodes advertising the IP is greater than the ECMP group limit of the router, the router will use less nodes than the ones advertising the IP.

For example, if the external traffic policy is set to **local** and the router has an ECMP group limit set to 16 and the pods implementing a LoadBalancer service are deployed on 30 nodes, this would result in pods deployed on 14 nodes not receiving any traffic. In this situation, it would be preferable to set the external traffic policy for the service to **cluster**.

35.1.5. MetalLB concepts for layer 2 mode

In layer 2 mode, the **speaker** pod on one node announces the external IP address for a service to the host network. From a network perspective, the node appears to have multiple IP addresses assigned to a network interface.
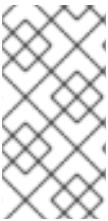
NOTE

In layer 2 mode, MetalLB relies on ARP and NDP. These protocols implement local address resolution within a specific subnet. In this context, the client must be able to reach the VIP assigned by MetalLB that exists on the same subnet as the nodes announcing the service in order for MetalLB to work.

The **speaker** pod responds to ARP requests for IPv4 services and NDP requests for IPv6.

In layer 2 mode, all traffic for a service IP address is routed through one node. After traffic enters the node, the service proxy for the CNI network provider distributes the traffic to all the pods for the service.

Because all traffic for a service enters through a single node in layer 2 mode, in a strict sense, MetalLB does not implement a load balancer for layer 2. Rather, MetalLB implements a failover mechanism for layer 2 so that when a **speaker** pod becomes unavailable, a **speaker** pod on a different node can announce the service IP address.

When a node becomes unavailable, failover is automatic. The **speaker** pods on the other nodes detect that a node is unavailable and a new **speaker** pod and node take ownership of the service IP address from the failed node.
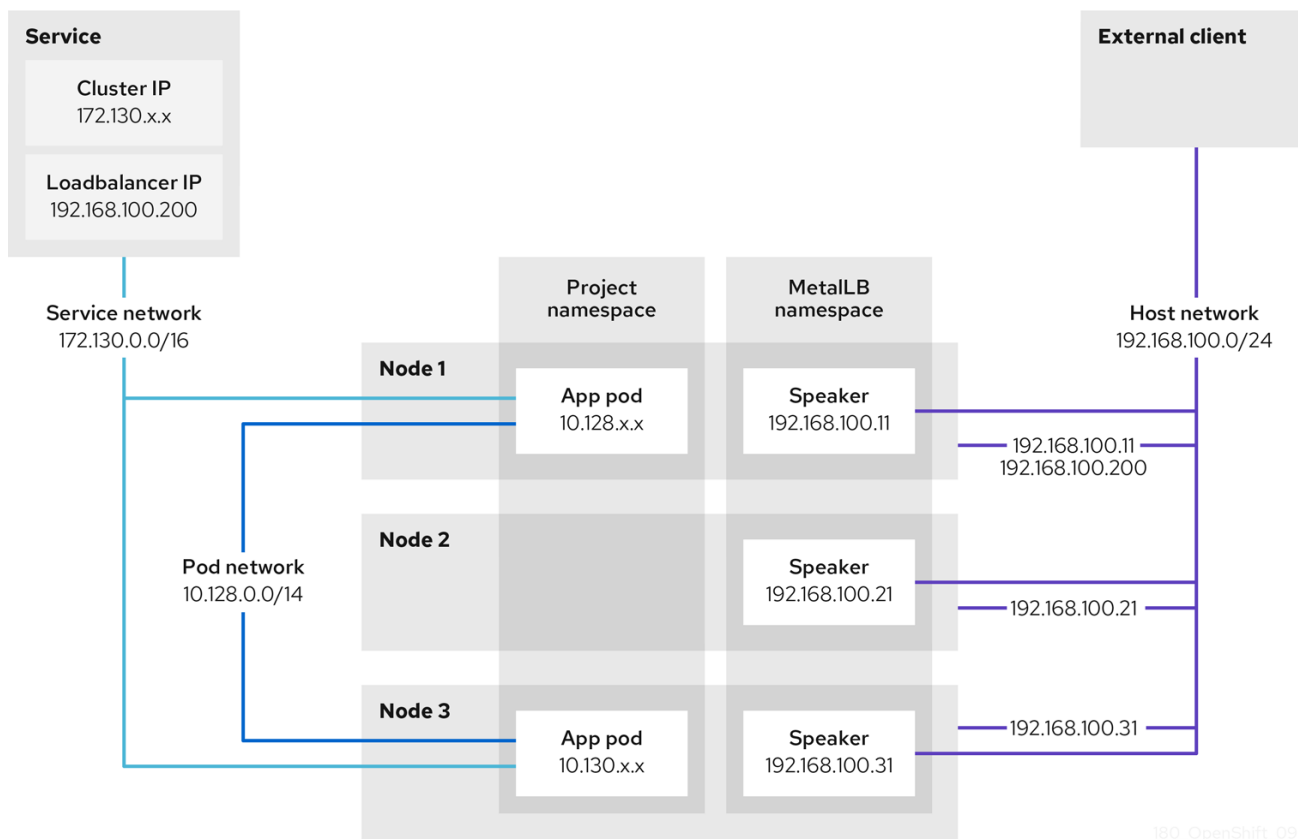
The preceding graphic shows the following concepts related to MetalLB:

- An application is available through a service that has a cluster IP on the **172.130.0.0/16** subnet. That IP address is accessible from inside the cluster. The service also has an external IP address that MetalLB assigned to the service, **192.168.100.200**.

- Nodes 1 and 3 have a pod for the application.

- The **speaker** daemon set runs a pod on each node. The MetalLB Operator starts these pods.

- Each **speaker** pod is a host-networked pod. The IP address for the pod is identical to the IP address for the node on the host network.

- The **speaker** pod on node 1 uses ARP to announce the external IP address for the service, **192.168.100.200**. The **speaker** pod that announces the external IP address must be on the same node as an endpoint for the service and the endpoint must be in the **Ready** condition.

- Client traffic is routed to the host network and connects to the **192.168.100.200** IP address. After traffic enters the node, the service proxy sends the traffic to the application pod on the same node or another node according to the external traffic policy that you set for the service.

    - If the external traffic policy for the service is set to **cluster**, the node that advertises the **192.168.100.200** load balancer IP address is selected from the nodes where a **speaker** pod is running. Only that node can receive traffic for the service.

    - If the external traffic policy for the service is set to **local**, the node that advertises the **192.168.100.200** load balancer IP address is selected from the nodes where a **speaker** pod is running and at least an endpoint of the service. Only that node can receive traffic for the service. In the preceding graphic, either node 1 or 3 would advertise **192.168.100.200**.

- If node 1 becomes unavailable, the external IP address fails over to another node. On another

node that has an instance of the application pod and service endpoint, the **speaker** pod begins to announce the external IP address, **192.168.100.200** and the new node receives the client traffic. In the diagram, the only candidate is node 3.

35.1.6. MetalLB concepts for BGP mode

In BGP mode, by default each **speaker** pod advertises the load balancer IP address for a service to each BGP peer. It is also possible to advertise the IPs coming from a given pool to a specific set of peers by adding an optional list of BGP peers. BGP peers are commonly network routers that are configured to use the BGP protocol. When a router receives traffic for the load balancer IP address, the router picks one of the nodes with a **speaker** pod that advertised the IP address. The router sends the traffic to that node. After traffic enters the node, the service proxy for the CNI network plugin distributes the traffic to all the pods for the service.

The directly-connected router on the same layer 2 network segment as the cluster nodes can be configured as a BGP peer. If the directly-connected router is not configured as a BGP peer, you need to configure your network so that packets for load balancer IP addresses are routed between the BGP peers and the cluster nodes that run the **speaker** pods.

Each time a router receives new traffic for the load balancer IP address, it creates a new connection to a node. Each router manufacturer has an implementation-specific algorithm for choosing which node to initiate the connection with. However, the algorithms commonly are designed to distribute traffic across the available nodes for the purpose of balancing the network load.

If a node becomes unavailable, the router initiates a new connection with another node that has a **speaker** pod that advertises the load balancer IP address.

Figure 35.1. MetalLB topology diagram for BGP mode
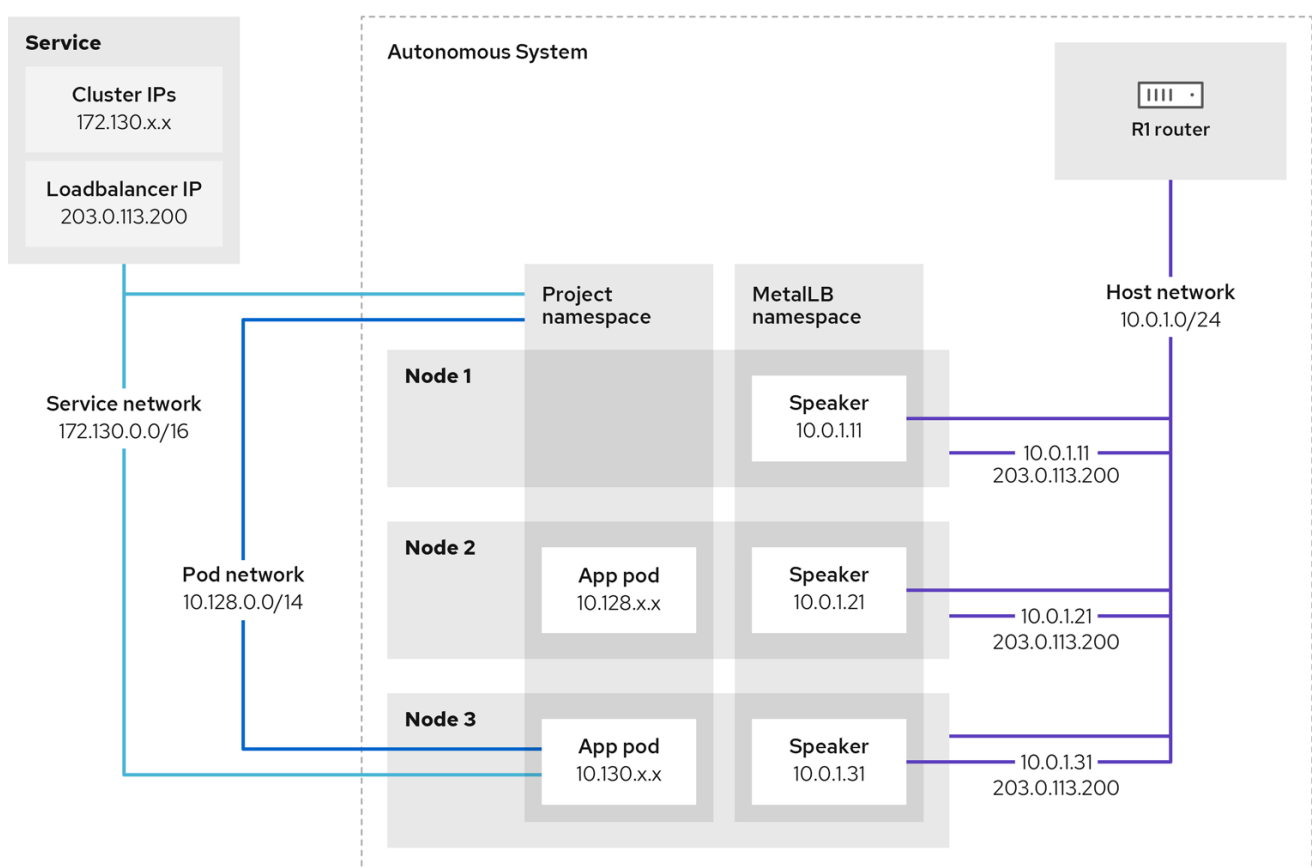
The preceding graphic shows the following concepts related to MetalLB:

- An application is available through a service that has an IPv4 cluster IP on the **172.130.0.0/16** subnet. That IP address is accessible from inside the cluster. The service also has an external IP address that MetalLB assigned to the service, **203.0.113.200**.

- Nodes 2 and 3 have a pod for the application.

- The **speaker** daemon set runs a pod on each node. The MetalLB Operator starts these pods. You can configure MetalLB to specify which nodes run the **speaker** pods.

- Each **speaker** pod is a host-networked pod. The IP address for the pod is identical to the IP address for the node on the host network.

- Each **speaker** pod starts a BGP session with all BGP peers and advertises the load balancer IP addresses or aggregated routes to the BGP peers. The **speaker** pods advertise that they are part of Autonomous System 65010. The diagram shows a router, R1, as a BGP peer within the same Autonomous System. However, you can configure MetalLB to start BGP sessions with peers that belong to other Autonomous Systems.

- All the nodes with a **speaker** pod that advertises the load balancer IP address can receive traffic for the service.

  - If the external traffic policy for the service is set to **cluster**, all the nodes where a speaker pod is running advertise the **203.0.113.200** load balancer IP address and all the nodes with a **speaker** pod can receive traffic for the service. The host prefix is advertised to the router peer only if the external traffic policy is set to cluster.

  - If the external traffic policy for the service is set to **local**, then all the nodes where a **speaker** pod is running and at least an endpoint of the service is running can advertise the **203.0.113.200** load balancer IP address. Only those nodes can receive traffic for the service. In the preceding graphic, nodes 2 and 3 would advertise **203.0.113.200**.

- You can configure MetalLB to control which **speaker** pods start BGP sessions with specific BGP peers by specifying a node selector when you add a BGP peer custom resource.

- Any routers, such as R1, that are configured to use BGP can be set as BGP peers.

- Client traffic is routed to one of the nodes on the host network. After traffic enters the node, the service proxy sends the traffic to the application pod on the same node or another node according to the external traffic policy that you set for the service.

- If a node becomes unavailable, the router detects the failure and initiates a new connection with another node. You can configure MetalLB to use a Bidirectional Forwarding Detection (BFD) profile for BGP peers. BFD provides faster link failure detection so that routers can initiate new connections earlier than without BFD.

35.1.7. Limitations and restrictions

35.1.7.1. Infrastructure considerations for MetalLB

MetalLB is primarily useful for on-premise, bare metal installations because these installations do not include a native load-balancer capability. In addition to bare metal installations, installations of OpenShift Container Platform on some infrastructures might not include a native load-balancer capability. For example, the following infrastructures can benefit from adding the MetalLB Operator:

- Bare metal

- VMware vSphere

- IBM Z and IBM® LinuxONE

- IBM Z and IBM® LinuxONE for Red Hat Enterprise Linux (RHEL) KVM

- IBM Power

MetalLB Operator and MetalLB are supported with the OpenShift SDN and OVN-Kubernetes network providers.

35.1.7.2. Limitations for layer 2 mode

35.1.7.2.1. Single-node bottleneck

MetalLB routes all traffic for a service through a single node, the node can become a bottleneck and limit performance.

Layer 2 mode limits the ingress bandwidth for your service to the bandwidth of a single node. This is a fundamental limitation of using ARP and NDP to direct traffic.

35.1.7.2.2. Slow failover performance

Failover between nodes depends on cooperation from the clients. When a failover occurs, MetalLB sends gratuitous ARP packets to notify clients that the MAC address associated with the service IP has changed.

Most client operating systems handle gratuitous ARP packets correctly and update their neighbor caches promptly. When clients update their caches quickly, failover completes within a few seconds. Clients typically fail over to a new node within 10 seconds. However, some client operating systems either do not handle gratuitous ARP packets at all or have outdated implementations that delay the cache update.

Recent versions of common operating systems such as Windows, macOS, and Linux implement layer 2 failover correctly. Issues with slow failover are not expected except for older and less common client operating systems.

To minimize the impact from a planned failover on outdated clients, keep the old node running for a few minutes after flipping leadership. The old node can continue to forward traffic for outdated clients until their caches refresh.

During an unplanned failover, the service IPs are unreachable until the outdated clients refresh their cache entries.

35.1.7.2.3. Additional Network and MetalLB cannot use same network

Using the same VLAN for both MetalLB and an additional network interface set up on a source pod might result in a connection failure. This occurs when both the MetalLB IP and the source pod reside on the same node.

To avoid connection failures, place the MetalLB IP in a different subnet from the one where the source pod resides. This configuration ensures that traffic from the source pod will take the default gateway. Consequently, the traffic can effectively reach its destination by using the OVN overlay network, ensuring that the connection functions as intended.

35.1.7.3. Limitations for BGP mode

35.1.7.3.1. Node failure can break all active connections

MetalLB shares a limitation that is common to BGP-based load balancing. When a BGP session terminates, such as when a node fails or when a **speaker** pod restarts, the session termination might result in resetting all active connections. End users can experience a **Connection reset by peer** message.

The consequence of a terminated BGP session is implementation-specific for each router manufacturer. However, you can anticipate that a change in the number of **speaker** pods affects the number of BGP sessions and that active connections with BGP peers will break.

To avoid or reduce the likelihood of a service interruption, you can specify a node selector when you add a BGP peer. By limiting the number of nodes that start BGP sessions, a fault on a node that does not have a BGP session has no affect on connections to the service.

35.1.7.3.2. Support for a single ASN and a single router ID only

When you add a BGP peer custom resource, you specify the **spec.myASN** field to identify the Autonomous System Number (ASN) that MetalLB belongs to. OpenShift Container Platform uses an implementation of BGP with MetalLB that requires MetalLB to belong to a single ASN. If you attempt to add a BGP peer and specify a different value for **spec.myASN** than an existing BGP peer custom resource, you receive an error.

Similarly, when you add a BGP peer custom resource, the **spec.routerID** field is optional. If you specify a value for this field, you must specify the same value for all other BGP peer custom resources that you add.

The limitation to support a single ASN and single router ID is a difference with the community-supported implementation of MetalLB.

35.1.8. Additional resources

- [Comparison: Fault tolerant access to external IP addresses](#)

- [Removing IP failover](#)

- [Deployment specifications for MetalLB](#)

35.2. INSTALLING THE METALLB OPERATOR

As a cluster administrator, you can add the MetallB Operator so that the Operator can manage the lifecycle for an instance of MetalLB on your cluster.

MetalLB and IP failover are incompatible. If you configured IP failover for your cluster, perform the steps to [remove IP failover](#) before you install the Operator.

35.2.1. Installing the MetalLB Operator from the OperatorHub using the web console

As a cluster administrator, you can install the MetalLB Operator by using the OpenShift Container Platform web console.

Prerequisites

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

Procedure

1. In the OpenShift Container Platform web console, navigate to **Operators → OperatorHub**.

2. Type a keyword into the **Filter by keyword** box or scroll to find the Operator you want. For example, type **metallb** to find the MetalLB Operator.
   You can also filter options by **Infrastructure Features**. For example, select **Disconnected** if you want to see Operators that work in disconnected environments, also known as restricted network environments.

3. On the **Install Operator** page, accept the defaults and click **Install**.

Verification

1. To confirm that the installation is successful:

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

   b. Check that the Operator is installed in the **openshift-operators** namespace and that its status is **Succeeded**.

2. If the Operator is not installed successfully, check the status of the Operator and review the logs:

   a. Navigate to the **Operators → Installed Operators** page and inspect the **Status** column for any errors or failures.

   b. Navigate to the **Workloads → Pods** page and check the logs in any pods in the **openshift-operators** project that are reporting issues.

35.2.2. Installing from OperatorHub using the CLI

Instead of using the OpenShift Container Platform web console, you can install an Operator from OperatorHub using the CLI. You can use the OpenShift CLI (**oc**) to install the MetalLB Operator.

It is recommended that when using the CLI you install the Operator in the **metallb-system** namespace.

Prerequisites

- A cluster installed on bare-metal hardware.

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

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

Procedure

1. Create a namespace for the MetalLB Operator by entering the following command:

   ```
   $ cat << EOF | oc apply -f -
   apiVersion: v1
   kind: Namespace
   ```

```
metadata:
  name: metallb-system
EOF
```

2. Create an Operator group custom resource (CR) in the namespace:

```
$ cat << EOF | oc apply -f -
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: metallb-operator
  namespace: metallb-system
EOF
```

3. Confirm the Operator group is installed in the namespace:

```
$ oc get operatorgroup -n metallb-system
```

Example output

```
NAME             AGE
metallb-operator   14m
```

4. Create a **Subscription** CR:

   a. Define the **Subscription** CR and save the YAML file, for example, **metallb-sub.yaml**:

   ```
   apiVersion: operators.coreos.com/v1alpha1
   kind: Subscription
   metadata:
     name: metallb-operator-sub
     namespace: metallb-system
   spec:
     channel: stable
     name: metallb-operator
     source: redhat-operators ❶
     sourceNamespace: openshift-marketplace
   ```

   ❶ You must specify the **redhat-operators** value.

   b. To create the **Subscription** CR, run the following command:

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

5. Optional: To ensure BGP and BFD metrics appear in Prometheus, you can label the namespace as in the following command:

```
$ oc label ns metallb-system "openshift.io/cluster-monitoring=true"
```

Verification

The verification steps assume the MetalLB Operator is installed in the **metallb-system** namespace.

1. Confirm the install plan is in the namespace:

   ```
   $ oc get installplan -n metallb-system
   ```

   Example output

   ```
   NAME         CSV                             APPROVAL   APPROVED
   install-wzg94   metallb-operator.4.13.0-nnnnnnnnnnnn   Automatic   true
   ```

   > **NOTE**
   >
   > Installation of the Operator might take a few seconds.

2. To verify that the Operator is installed, enter the following command:

   ```
   $ oc get clusterserviceversion -n metallb-system \
     -o custom-columns=Name:.metadata.name,Phase:.status.phase
   ```

   Example output

   ```
   Name                        Phase
   metallb-operator.4.13.0-nnnnnnnnnnnn   Succeeded
   ```

35.2.3. Starting MetalLB on your cluster

After you install the Operator, you need to configure a single instance of a MetalLB custom resource. After you configure the custom resource, the Operator starts MetalLB on your cluster.

Prerequisites

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

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

- Install the MetalLB Operator.

Procedure

This procedure assumes the MetalLB Operator is installed in the **metallb-system** namespace. If you installed using the web console substitute **openshift-operators** for the namespace.

1. Create a single instance of a MetalLB custom resource:

   ```
   $ cat << EOF | oc apply -f -
   apiVersion: metallb.io/v1beta1
   kind: MetalLB
   metadata:
     name: metallb
     namespace: metallb-system
   EOF
   ```

Verification

Confirm that the deployment for the MetalLB controller and the daemon set for the MetalLB speaker are running.

1. Verify that the deployment for the controller is running:

```
$ oc get deployment -n metallb-system controller
```

Example output

```
NAME        READY   UP-TO-DATE  AVAILABLE  AGE
controller  1/1     1           1          11m
```

2. Verify that the daemon set for the speaker is running:

```
$ oc get daemonset -n metallb-system speaker
```

Example output

```
NAME     DESIRED  CURRENT  READY  UP-TO-DATE  AVAILABLE  NODE
SELECTOR          AGE
speaker  6        6        6      6           6          kubernetes.io/os=linux  18m
```

The example output indicates 6 speaker pods. The number of speaker pods in your cluster might differ from the example output. Make sure the output indicates one pod for each node in your cluster.

35.2.4. Deployment specifications for MetalLB

When you start an instance of MetalLB using the **MetalLB** custom resource, you can configure deployment specifications in the **MetalLB** custom resource to manage how the **controller** or **speaker** pods deploy and run in your cluster. Use these deployment specifications to manage the following tasks:

- Select nodes for MetalLB pod deployment.

- Manage scheduling by using pod priority and pod affinity.

- Assign CPU limits for MetalLB pods.

- Assign a container RuntimeClass for MetalLB pods.

- Assign metadata for MetalLB pods.

35.2.4.1. Limit speaker pods to specific nodes

By default, when you start MetalLB with the MetalLB Operator, the Operator starts an instance of a **speaker** pod on each node in the cluster. Only the nodes with a **speaker** pod can advertise a load balancer IP address. You can configure the **MetalLB** custom resource with a node selector to specify which nodes run the **speaker** pods.

The most common reason to limit the **speaker** pods to specific nodes is to ensure that only nodes with network interfaces on specific networks advertise load balancer IP addresses. Only the nodes with a running **speaker** pod are advertised as destinations of the load balancer IP address.

If you limit the **speaker** pods to specific nodes and specify **local** for the external traffic policy of a service, then you must ensure that the application pods for the service are deployed to the same nodes.

Example configuration to limit speaker pods to worker nodes

```
apiVersion: metallb.io/v1beta1
kind: MetalLB
metadata:
  name: metallb
  namespace: metallb-system
spec:
  nodeSelector:  <.>
    node-role.kubernetes.io/worker: ""
  speakerTolerations:   <.>
  - key: "Example"
    operator: "Exists"
    effect: "NoExecute"
```

<.> The example configuration specifies to assign the speaker pods to worker nodes, but you can specify labels that you assigned to nodes or any valid node selector. <.> In this example configuration, the pod that this toleration is attached to tolerates any taint that matches the **key** value and **effect** value using the **operator**.

After you apply a manifest with the **spec.nodeSelector** field, you can check the number of pods that the Operator deployed with the **oc get daemonset -n metallb-system speaker** command. Similarly, you can display the nodes that match your labels with a command like **oc get nodes -l node-role.kubernetes.io/worker=**.

You can optionally allow the node to control which speaker pods should, or should not, be scheduled on them by using affinity rules. You can also limit these pods by applying a list of tolerations. For more information about affinity rules, taints, and tolerations, see the additional resources.

35.2.4.2. Configuring a container runtime class in a MetalLB deployment

You can optionally assign a container runtime class to **controller** and **speaker** pods by configuring the MetalLB custom resource. For example, for Windows workloads, you can assign a Windows runtime class to the pod, which uses this runtime class for all containers in the pod.

Prerequisites

- You are logged in as a user with **cluster-admin** privileges.

- You have installed the MetalLB Operator.

Procedure

1. Create a **RuntimeClass** custom resource, such as **myRuntimeClass.yaml**, to define your runtime class:

```
apiVersion: node.k8s.io/v1
kind: RuntimeClass
metadata:
  name: myclass
handler: myconfiguration
```

2. Apply the **RuntimeClass** custom resource configuration:

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

3. Create a **MetalLB** custom resource, such as **MetalLBRuntime.yaml**, to specify the **runtimeClassName** value:

```
apiVersion: metallb.io/v1beta1
kind: MetalLB
metadata:
  name: metallb
  namespace: metallb-system
spec:
  logLevel: debug
  controllerConfig:
    runtimeClassName: myclass
    annotations:     ❶
      controller: demo
  speakerConfig:
    runtimeClassName: myclass
    annotations:     ❷
      speaker: demo
```

❶ ❷ This example uses **annotations** to add metadata such as build release information or GitHub pull request information. You can populate annotations with characters that are not permitted in labels. However, you cannot use annotations to identify or select objects.

4. Apply the **MetalLB** custom resource configuration:

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

Verification

- To view the container runtime for a pod, run the following command:

```
$ oc get pod -o custom-
columns=NAME:metadata.name,STATUS:.status.phase,RUNTIME_CLASS:.spec.runtimeClass
Name
```

35.2.4.3. Configuring pod priority and pod affinity in a MetalLB deployment

You can optionally assign pod priority and pod affinity rules to **controller** and **speaker** pods by configuring the **MetalLB** custom resource. The pod priority indicates the relative importance of a pod on a node and schedules the pod based on this priority. Set a high priority on your **controller** or **speaker** pod to ensure scheduling priority over other pods on the node.

Pod affinity manages relationships among pods. Assign pod affinity to the **controller** or **speaker** pods to control on what node the scheduler places the pod in the context of pod relationships. For example, you can use pod affinity rules to ensure that certain pods are located on the same node or nodes, which can help improve network communication and reduce latency between those components.

Prerequisites

- You are logged in as a user with **cluster-admin** privileges.

- You have installed the MetalLB Operator.

- You have started the MetalLB Operator on your cluster.

Procedure

1. Create a **PriorityClass** custom resource, such as **myPriorityClass.yaml**, to configure the priority level. This example defines a **PriorityClass** named **high-priority** with a value of **1000000**. Pods that are assigned this priority class are considered higher priority during scheduling compared to pods with lower priority classes:

   ```
   apiVersion: scheduling.k8s.io/v1
   kind: PriorityClass
   metadata:
     name: high-priority
   value: 1000000
   ```

2. Apply the **PriorityClass** custom resource configuration:

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

3. Create a **MetalLB** custom resource, such as **MetalLBPodConfig.yaml**, to specify the **priorityClassName** and **podAffinity** values:

   ```
   apiVersion: metallb.io/v1beta1
   kind: MetalLB
   metadata:
     name: metallb
     namespace: metallb-system
   spec:
     logLevel: debug
     controllerConfig:
       priorityClassName: high-priority 1
       affinity:
         podAffinity: 2
           requiredDuringSchedulingIgnoredDuringExecution:
           - labelSelector:
               matchLabels:
                 app: metallb
             topologyKey: kubernetes.io/hostname
     speakerConfig:
       priorityClassName: high-priority
       affinity:
         podAffinity:
           requiredDuringSchedulingIgnoredDuringExecution:
           - labelSelector:
               matchLabels:
                 app: metallb
             topologyKey: kubernetes.io/hostname
   ```

   1 Specifies the priority class for the MetalLB controller pods. In this case, it is set to **high-priority**.

2 Specifies that you are configuring pod affinity rules. These rules dictate how pods are scheduled in relation to other pods or nodes. This configuration instructs the scheduler to

4. Apply the **MetalLB** custom resource configuration:

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

Verification

- To view the priority class that you assigned to pods in the **metallb-system** namespace, run the following command:

```
$ oc get pods -n metallb-system -o custom-
columns=NAME:.metadata.name,PRIORITY:.spec.priorityClassName
```

Example output

```
NAME                                           PRIORITY
controller-584f5c8cd8-5zbvg                    high-priority
metallb-operator-controller-manager-9c8d9985-szkqg   <none>
metallb-operator-webhook-server-c895594d4-shjgx      <none>
speaker-dddf7                                  high-priority
```

- To verify that the scheduler placed pods according to pod affinity rules, view the metadata for the pod's node or nodes by running the following command:

```
$ oc get pod -o=custom-columns=NODE:.spec.nodeName,NAME:.metadata.name -n
metallb-system
```

35.2.4.4. Configuring pod CPU limits in a MetalLB deployment

You can optionally assign pod CPU limits to **controller** and **speaker** pods by configuring the **MetalLB** custom resource. Defining CPU limits for the **controller** or **speaker** pods helps you to manage compute resources on the node. This ensures all pods on the node have the necessary compute resources to manage workloads and cluster housekeeping.

Prerequisites

- You are logged in as a user with **cluster-admin** privileges.

- You have installed the MetalLB Operator.

Procedure

1. Create a **MetalLB** custom resource file, such as **CPULimits.yaml**, to specify the **cpu** value for the **controller** and **speaker** pods:

```
apiVersion: metallb.io/v1beta1
kind: MetalLB
metadata:
  name: metallb
  namespace: metallb-system
```

```
spec:
  logLevel: debug
  controllerConfig:
    resources:
      limits:
        cpu: "200m"
  speakerConfig:
    resources:
      limits:
        cpu: "300m"
```

2. Apply the **MetalLB** custom resource configuration:

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

Verification

- To view compute resources for a pod, run the following command, replacing **<pod_name>** with your target pod:

```
$ oc describe pod <pod_name>
```

35.2.5. Additional resources

- [Placing pods on specific nodes using node selectors](#)

- [Understanding taints and tolerations](#)

- [Understanding pod priority](#)

- [Understanding pod affinity](#)

35.2.6. Next steps

- [Configuring MetalLB address pools](#)

35.3. UPGRADING THE METALLB

If you are currently running version 4.10 or an earlier version of the MetalLB Operator, please note that automatic updates to any version later than 4.10 do not work. Upgrading to a newer version from any version of the MetalLB Operator that is 4.11 or later is successful. For example, upgrading from version 4.12 to version 4.13 will occur smoothly.

A summary of the upgrade procedure for the MetalLB Operator from 4.10 and earlier is as follows:

1. Delete the installed MetalLB Operator version for example 4.10. Ensure that the namespace and the **metallb** custom resource are not removed.

2. Using the CLI, install the MetalLB Operator 4.13 in the same namespace where the previous version of the MetalLB Operator was installed.

> **NOTE**
>
> This procedure does not apply to automatic z-stream updates of the MetalLB Operator, which follow the standard straightforward method.

For detailed steps to upgrade the MetalLB Operator from 4.10 and earlier, see the guidance that follows. As a cluster administrator, start the upgrade process by deleting the MetalLB Operator by using the OpenShift CLI (**oc**) or the web console.

35.3.1. Deleting the MetalLB Operator from a cluster using the web console

Cluster administrators can delete installed Operators from a selected namespace by using the web console.

Prerequisites

- Access to an OpenShift Container Platform cluster web console using an account with **cluster-admin** permissions.

Procedure

1. Navigate to the **Operators → Installed Operators** page.

2. Search for the MetalLB Operator. Then, click on it.

3. On the right side of the **Operator Details** page, select **Uninstall Operator** from the **Actions** drop-down menu.
   An **Uninstall Operator?** dialog box is displayed.

4. Select **Uninstall** to remove the Operator, Operator deployments, and pods. Following this action, the Operator stops running and no longer receives updates.

   > **NOTE**
   >
   > This action does not remove resources managed by the Operator, including custom resource definitions (CRDs) and custom resources (CRs). Dashboards and navigation items enabled by the web console and off-cluster resources that continue to run might need manual clean up. To remove these after uninstalling the Operator, you might need to manually delete the Operator CRDs.

35.3.2. Deleting MetalLB Operator from a cluster using the CLI

Cluster administrators can delete installed Operators from a selected namespace by using the CLI.

Prerequisites

- Access to an OpenShift Container Platform cluster using an account with **cluster-admin** permissions.

- **oc** command installed on workstation.

Procedure

1. Check the current version of the subscribed MetalLB Operator in the **currentCSV** field:

```
$ oc get subscription metallb-operator -n metallb-system -o yaml | grep currentCSV
```

Example output

```
currentCSV: metallb-operator.4.10.0-202207051316
```

2. Delete the subscription:

```
$ oc delete subscription metallb-operator -n metallb-system
```

Example output

```
subscription.operators.coreos.com "metallb-operator" deleted
```

3. Delete the CSV for the Operator in the target namespace using the **currentCSV** value from the previous step:

```
$ oc delete clusterserviceversion metallb-operator.4.10.0-202207051316 -n metallb-system
```

Example output

```
clusterserviceversion.operators.coreos.com "metallb-operator.4.10.0-202207051316" deleted
```

35.3.3. Editing the MetalLB Operator Operator group

When upgrading from any MetalLB Operator version up to and including 4.10 to 4.11 and later, remove **spec.targetNamespaces** from the Operator group custom resource (CR). You must remove the spec regardless of whether you used the web console or the CLI to delete the MetalLB Operator.

> **NOTE**
>
> The MetalLB Operator version 4.11 or later only supports the **AllNamespaces** install mode, whereas 4.10 or earlier versions support **OwnNamespace** or **SingleNamespace** modes.

Prerequisites

- You have access to an OpenShift Container Platform cluster with **cluster-admin** permissions.

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

Procedure

1. List the Operator groups in the **metallb-system** namespace by running the following command:

```
$ oc get operatorgroup -n metallb-system
```

Example output

```
NAME                AGE
metallb-system-7jc66   85m
```

2. Verify that the **spec.targetNamespaces** is present in the Operator group CR associated with the **metallb-system** namespace by running the following command:

```
$ oc get operatorgroup metallb-system-7jc66 -n metallb-system -o yaml
```

Example output

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  annotations:
    olm.providedAPIs: ""
  creationTimestamp: "2023-10-25T09:42:49Z"
  generateName: metallb-system-
  generation: 1
  name: metallb-system-7jc66
  namespace: metallb-system
  resourceVersion: "25027"
  uid: f5f644a0-eef8-4e31-a306-e2bbcfaffab3
spec:
  targetNamespaces:
  - metallb-system
  upgradeStrategy: Default
status:
  lastUpdated: "2023-10-25T09:42:49Z"
  namespaces:
  - metallb-system
```

3. Edit the Operator group and remove the **targetNamespaces** and **metallb-system** present under the **spec** section by running the following command:

```
$ oc edit n metallb-system
```

Example output

```
operatorgroup.operators.coreos.com/metallb-system-7jc66 edited
```

4. Verify the **spec.targetNamespaces** is removed from the Operator group custom resource associated with the **metallb-system** namespace by running the following command:

```
$ oc get operatorgroup metallb-system-7jc66 -n metallb-system -o yaml
```

Example output

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  annotations:
    olm.providedAPIs: ""
  creationTimestamp: "2023-10-25T09:42:49Z"
  generateName: metallb-system-
  generation: 2
  name: metallb-system-7jc66
```

```
  namespace: metallb-system
  resourceVersion: "61658"
  uid: f5f644a0-eef8-4e31-a306-e2bbcfaffab3
spec:
  upgradeStrategy: Default
status:
  lastUpdated: "2023-10-25T14:31:30Z"
  namespaces:
  - ""
```

35.3.4. Upgrading the MetalLB Operator

Prerequisites

- Access the cluster as a user with the **cluster-admin** role.

Procedure

1. Verify that the **metallb-system** namespace still exists:

   ```
   $ oc get namespaces | grep metallb-system
   ```

   Example output

   ```
   metallb-system                       Active   31m
   ```

2. Verify the **metallb** custom resource still exists:

   ```
   $ oc get metallb -n metallb-system
   ```

   Example output

   ```
   NAME     AGE
   metallb   33m
   ```

3. Follow the guidance in "Installing from OperatorHub using the CLI" to install the latest 4.13 version of the MetalLB Operator.
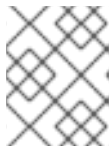
   > **NOTE**
   >
   > When installing the latest 4.13 version of the MetalLB Operator, you must install the Operator to the same namespace it was previously installed to.

4. Verify the upgraded version of the Operator is now the 4.13 version.

   ```
   $ oc get csv -n metallb-system
   ```

   Example output

```
NAME                              DISPLAY       VERSION            REPLACES   PHASE
metallb-operator.4.13.0-202207051316   MetalLB Operator   4.13.0-202207051316
Succeeded
```

35.3.5. Additional resources

-

-

35.4. CONFIGURING METALLB ADDRESS POOLS

As a cluster administrator, you can add, modify, and delete address pools. The MetalLB Operator uses the address pool custom resources to set the IP addresses that MetalLB can assign to services. The namespace used in the examples assume the namespace is **metallb-system**.

35.4.1. About the IPAddressPool custom resource

> **NOTE**
>
> The address pool custom resource definition (CRD) and API documented in "Load balancing with MetalLB" in OpenShift Container Platform 4.10 can still be used in 4.13. However, the enhanced functionality associated with advertising an IP address from an **IPAddressPool** with layer 2 protocols, or the BGP protocol, is not supported when using the **AddressPool** CRD.

The fields for the **IPAddressPool** custom resource are described in the following tables.

Table 35.1. MetalLB IPAddressPool pool custom resource

Field	Type	Description
metadata.name	**string**	Specifies the name for the address pool. When you add a service, you can specify this pool name in the **metallb.universe.tf/address-pool** annotation to select an IP address from a specific pool. The names **doc-example**, **silver**, and **gold** are used throughout the documentation.
metadata.name space	**string**	Specifies the namespace for the address pool. Specify the same namespace that the MetalLB Operator uses.
metadata.label	**string**	Optional: Specifies the key value pair assigned to the **IPAddressPool**. This can be referenced by the **ipAddressPoolSelectors** in the **BGPAdvertisement** and **L2Advertisement** CRD to associate the **IPAddressPool** with the advertisement

Field	Type	Description
spec.addresses	**string**	Specifies a list of IP addresses for MetalLB Operator to assign to services. You can specify multiple ranges in a single pool; they will all share the same settings. Specify each range in CIDR notation or as starting and ending IP addresses separated with a hyphen.
spec.autoAssign	**boolean**	Optional: Specifies whether MetalLB automatically assigns IP addresses from this pool. Specify **false** if you want explicitly request an IP address from this pool with the **metallb.universe.tf/address-pool** annotation. The default value is **true**.
spec.avoidBuggyIPs	**boolean**	Optional: This ensures when enabled that IP addresses ending .0 and .255 are not allocated from the pool. The default value is **false**. Some older consumer network equipment mistakenly block IP addresses ending in .0 and .255.

You can assign IP addresses from an **IPAddressPool** to services and namespaces by configuring the **spec.serviceAllocation** specification.

Table 35.2. MetalLB IPAddressPool custom resource spec.serviceAllocation subfields

Field	Type	Description
priority	**int**	Optional: Defines the priority between IP address pools when more than one IP address pool matches a service or namespace. A lower number indicates a higher priority.
namespaces	**array (string)**	Optional: Specifies a list of namespaces that you can assign to IP addresses in an IP address pool.
namespaceSelectors	**array (LabelSelector)**	Optional: Specifies namespace labels that you can assign to IP addresses from an IP address pool by using label selectors in a list format.
serviceSelectors	**array (LabelSelector)**	Optional: Specifies service labels that you can assign to IP addresses from an address pool by using label selectors in a list format.

35.4.2. Configuring an address pool

As a cluster administrator, you can add address pools to your cluster to control the IP addresses that MetalLB can assign to load-balancer services.

Prerequisites

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

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

Procedure

1. Create a file, such as **ipaddresspool.yaml**, with content like the following example:

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  namespace: metallb-system
  name: doc-example
  labels: ❶
    zone: east
spec:
  addresses:
  - 203.0.113.1-203.0.113.10
  - 203.0.113.65-203.0.113.75
```

❶   This label assigned to the **IPAddressPool** can be referenced by the **ipAddressPoolSelectors** in the **BGPAdvertisement** CRD to associate the **IPAddressPool** with the advertisement.

2. Apply the configuration for the IP address pool:

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

Verification

- View the address pool:

```
$ oc describe -n metallb-system IPAddressPool doc-example
```

Example output

```
Name:         doc-example
Namespace:    metallb-system
Labels:       zone=east
Annotations:  <none>
API Version:  metallb.io/v1beta1
Kind:         IPAddressPool
Metadata:
  ...
Spec:
  Addresses:
    203.0.113.1-203.0.113.10
    203.0.113.65-203.0.113.75
  Auto Assign:  true
Events:         <none>
```

Confirm that the address pool name, such as **doc-example**, and the IP address ranges appear in the output.

35.4.3. Example address pool configurations

35.4.3.1. Example: IPv4 and CIDR ranges

You can specify a range of IP addresses in CIDR notation. You can combine CIDR notation with the notation that uses a hyphen to separate lower and upper bounds.

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: doc-example-cidr
  namespace: metallb-system
spec:
  addresses:
  - 192.168.100.0/24
  - 192.168.200.0/24
  - 192.168.255.1-192.168.255.5
```

35.4.3.2. Example: Reserve IP addresses

You can set the **autoAssign** field to **false** to prevent MetalLB from automatically assigning the IP addresses from the pool. When you add a service, you can request a specific IP address from the pool or you can specify the pool name in an annotation to request any IP address from the pool.

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: doc-example-reserved
  namespace: metallb-system
spec:
  addresses:
  - 10.0.100.0/28
  autoAssign: false
```

35.4.3.3. Example: IPv4 and IPv6 addresses

You can add address pools that use IPv4 and IPv6. You can specify multiple ranges in the **addresses** list, just like several IPv4 examples.

Whether the service is assigned a single IPv4 address, a single IPv6 address, or both is determined by how you add the service. The **spec.ipFamilies** and **spec.ipFamilyPolicy** fields control how IP addresses are assigned to the service.

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: doc-example-combined
  namespace: metallb-system
spec:
  addresses:
  - 10.0.100.0/28
  - 2002:2:2::1-2002:2:2::100
```

35.4.3.4. Example: Assign IP address pools to services or namespaces

You can assign IP addresses from an **IPAddressPool** to services and namespaces that you specify.

If you assign a service or namespace to more than one IP address pool, MetalLB uses an available IP address from the higher-priority IP address pool. If no IP addresses are available from the assigned IP address pools with a high priority, MetalLB uses available IP addresses from an IP address pool with lower priority or no priority.

> **NOTE**
>
> You can use the **matchLabels** label selector, the **matchExpressions** label selector, or both, for the **namespaceSelectors** and **serviceSelectors** specifications. This example demonstrates one label selector for each specification.

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: doc-example-service-allocation
  namespace: metallb-system
spec:
  addresses:
    - 192.168.20.0/24
  serviceAllocation:
    priority: 50 1
    namespaces: 2
      - namespace-a
      - namespace-b
    namespaceSelectors: 3
      - matchLabels:
          zone: east
    serviceSelectors: 4
      - matchExpressions:
        - key: security
          operator: In
          values:
          - S1
```

1 Assign a priority to the address pool. A lower number indicates a higher priority.

2 Assign one or more namespaces to the IP address pool in a list format.

3 Assign one or more namespace labels to the IP address pool by using label selectors in a list format.

4 Assign one or more service labels to the IP address pool by using label selectors in a list format.

35.4.4. Additional resources

- Configuring MetalLB with an L2 advertisement and label .

35.4.5. Next steps

- For BGP mode, see Configuring MetalLB BGP peers.

- Configuring services to use MetalLB .

35.5. ABOUT ADVERTISING FOR THE IP ADDRESS POOLS

You can configure MetalLB so that the IP address is advertised with layer 2 protocols, the BGP protocol, or both. With layer 2, MetalLB provides a fault-tolerant external IP address. With BGP, MetalLB provides fault-tolerance for the external IP address and load balancing.

MetalLB supports advertising using L2 and BGP for the same set of IP addresses.

MetalLB provides the flexibility to assign address pools to specific BGP peers effectively to a subset of nodes on the network. This allows for more complex configurations, for example facilitating the isolation of nodes or the segmentation of the network.

35.5.1. About the BGPAdvertisement custom resource

The fields for the **BGPAdvertisements** object are defined in the following table:

Table 35.3. BGPAdvertisements configuration

Field	Type	Description
metadata.name	**string**	Specifies the name for the BGP advertisement.
metadata.name space	**string**	Specifies the namespace for the BGP advertisement. Specify the same namespace that the MetalLB Operator uses.
spec.aggregatio nLength	**integer**	Optional: Specifies the number of bits to include in a 32-bit CIDR mask. To aggregate the routes that the speaker advertises to BGP peers, the mask is applied to the routes for several service IP addresses and the speaker advertises the aggregated route. For example, with an aggregation length of **24**, the speaker can aggregate several **10.0.1.x/32** service IP addresses and advertise a single **10.0.1.0/24** route.
spec.aggregatio nLengthV6	**integer**	Optional: Specifies the number of bits to include in a 128-bit CIDR mask. For example, with an aggregation length of **124**, the speaker can aggregate several **fc00:f853:0ccd:e799::x/128** service IP addresses and advertise a single **fc00:f853:0ccd:e799::0/124** route.

Field	Type	Description
spec.communities	**string**	Optional: Specifies one or more BGP communities. Each community is specified as two 16-bit values separated by the colon character. Well-known communities must be specified as 16-bit values: • **NO_EXPORT**: 65535:65281 • **NO_ADVERTISE**: 65535:65282 • **NO_EXPORT_SUBCONFED**: 65535:65283 **NOTE** You can also use community objects that are created along with the strings.
spec.localPref	**integer**	Optional: Specifies the local preference for this advertisement. This BGP attribute applies to BGP sessions within the Autonomous System.
spec.ipAddressPools	**string**	Optional: The list of **IPAddressPools** to advertise with this advertisement, selected by name.
spec.ipAddressPoolSelectors	**string**	Optional: A selector for the **IPAddressPools** that gets advertised with this advertisement. This is for associating the **IPAddressPool** to the advertisement based on the label assigned to the **IPAddressPool** instead of the name itself. If no **IPAddressPool** is selected by this or by the list, the advertisement is applied to all the **IPAddressPools**.
spec.nodeSelectors	**string**	Optional: **NodeSelectors** allows to limit the nodes to announce as next hops for the load balancer IP. When empty, all the nodes are announced as next hops.
spec.peers	**string**	Optional: Peers limits the BGP peer to advertise the IPs of the selected pools to. When empty, the load balancer IP is announced to all the BGP peers configured.

35.5.2. Configuring MetalLB with a BGP advertisement and a basic use case

Configure MetalLB as follows so that the peer BGP routers receive one **203.0.113.200/32** route and one **fc00:f853:ccd:e799::1/128** route for each load-balancer IP address that MetalLB assigns to a service. Because the **localPref** and **communities** fields are not specified, the routes are advertised with **localPref** set to zero and no BGP communities.

35.5.2.1. Example: Advertise a basic address pool configuration with BGP

Configure MetalLB as follows so that the **IPAddressPool** is advertised with the BGP protocol.

Prerequisites

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

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

Procedure

1. Create an IP address pool.

   a. Create a file, such as **ipaddresspool.yaml**, with content like the following example:

   ```
   apiVersion: metallb.io/v1beta1
   kind: IPAddressPool
   metadata:
     namespace: metallb-system
     name: doc-example-bgp-basic
   spec:
     addresses:
       - 203.0.113.200/30
       - fc00:f853:ccd:e799::/124
   ```

   b. Apply the configuration for the IP address pool:

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

2. Create a BGP advertisement.

   a. Create a file, such as **bgpadvertisement.yaml**, with content like the following example:

   ```
   apiVersion: metallb.io/v1beta1
   kind: BGPAdvertisement
   metadata:
     name: bgpadvertisement-basic
     namespace: metallb-system
   spec:
     ipAddressPools:
     - doc-example-bgp-basic
   ```

   b. Apply the configuration:

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

35.5.3. Configuring MetalLB with a BGP advertisement and an advanced use case

Configure MetalLB as follows so that MetalLB assigns IP addresses to load-balancer services in the ranges between **203.0.113.200** and **203.0.113.203** and between **fc00:f853:ccd:e799::0** and **fc00:f853:ccd:e799::f**.

To explain the two BGP advertisements, consider an instance when MetalLB assigns the IP address of **203.0.113.200** to a service. With that IP address as an example, the speaker advertises two routes to BGP peers:

- **203.0.113.200/32**, with **localPref** set to **100** and the community set to the numeric value of the **NO_ADVERTISE** community. This specification indicates to the peer routers that they can use this route but they should not propagate information about this route to BGP peers.

- **203.0.113.200/30**, aggregates the load-balancer IP addresses assigned by MetalLB into a single route. MetalLB advertises the aggregated route to BGP peers with the community attribute set to **8000:800**. BGP peers propagate the **203.0.113.200/30** route to other BGP peers. When traffic is routed to a node with a speaker, the **203.0.113.200/32** route is used to forward the traffic into the cluster and to a pod that is associated with the service.

As you add more services and MetalLB assigns more load-balancer IP addresses from the pool, peer routers receive one local route, **203.0.113.20x/32**, for each service, as well as the **203.0.113.200/30** aggregate route. Each service that you add generates the **/30** route, but MetalLB deduplicates the routes to one BGP advertisement before communicating with peer routers.

35.5.3.1. Example: Advertise an advanced address pool configuration with BGP

Configure MetalLB as follows so that the **IPAddressPool** is advertised with the BGP protocol.

Prerequisites

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

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

Procedure

1. Create an IP address pool.

   a. Create a file, such as **ipaddresspool.yaml**, with content like the following example:

   ```
   apiVersion: metallb.io/v1beta1
   kind: IPAddressPool
   metadata:
     namespace: metallb-system
     name: doc-example-bgp-adv
     labels:
       zone: east
   spec:
     addresses:
       - 203.0.113.200/30
       - fc00:f853:ccd:e799::/124
     autoAssign: false
   ```

   b. Apply the configuration for the IP address pool:

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

2. Create a BGP advertisement.

   a. Create a file, such as **bgpadvertisement1.yaml**, with content like the following example:

   ```
   apiVersion: metallb.io/v1beta1
   kind: BGPAdvertisement
   metadata:
   ```

```
  name: bgpadvertisement-adv-1
  namespace: metallb-system
spec:
 ipAddressPools:
   - doc-example-bgp-adv
 communities:
   - 65535:65282
 aggregationLength: 32
 localPref: 100
```

b. Apply the configuration:

```
$ oc apply -f bgpadvertisement1.yaml
```

c. Create a file, such as **bgpadvertisement2.yaml**, with content like the following example:

```
apiVersion: metallb.io/v1beta1
kind: BGPAdvertisement
metadata:
  name: bgpadvertisement-adv-2
  namespace: metallb-system
spec:
 ipAddressPools:
   - doc-example-bgp-adv
 communities:
   - 8000:800
 aggregationLength: 30
 aggregationLengthV6: 124
```

d. Apply the configuration:

```
$ oc apply -f bgpadvertisement2.yaml
```

35.5.4. Advertising an IP address pool from a subset of nodes

To advertise an IP address from an IP addresses pool, from a specific set of nodes only, use the **.spec.nodeSelector** specification in the BGPAdvertisement custom resource. This specification associates a pool of IP addresses with a set of nodes in the cluster. This is useful when you have nodes on different subnets in a cluster and you want to advertise an IP addresses from an address pool from a specific subnet, for example a public-facing subnet only.

Prerequisites

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

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

Procedure

1. Create an IP address pool by using a custom resource:

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
```

```
  namespace: metallb-system
  name: pool1
spec:
 addresses:
   - 4.4.4.100-4.4.4.200
   - 2001:100:4::200-2001:100:4::400
```

2. Control which nodes in the cluster the IP address from **pool1** advertises from by defining the **.spec.nodeSelector** value in the BGPAdvertisement custom resource:

```
apiVersion: metallb.io/v1beta1
kind: BGPAdvertisement
metadata:
  name: example
spec:
 ipAddressPools:
 - pool1
 nodeSelector:
 - matchLabels:
     kubernetes.io/hostname: NodeA
 - matchLabels:
     kubernetes.io/hostname: NodeB
```

In this example, the IP address from **pool1** advertises from **NodeA** and **NodeB** only.

35.5.5. About the L2Advertisement custom resource

The fields for the **l2Advertisements** object are defined in the following table:

Table 35.4. L2 advertisements configuration

Field	Type	Description
metadata.name	**string**	Specifies the name for the L2 advertisement.
metadata.name space	**string**	Specifies the namespace for the L2 advertisement. Specify the same namespace that the MetalLB Operator uses.
spec.ipAddress Pools	**string**	Optional: The list of **IPAddressPools** to advertise with this advertisement, selected by name.
spec.ipAddress PoolSelectors	**string**	Optional: A selector for the **IPAddressPools** that gets advertised with this advertisement. This is for associating the **IPAddressPool** to the advertisement based on the label assigned to the **IPAddressPool** instead of the name itself. If no **IPAddressPool** is selected by this or by the list, the advertisement is applied to all the **IPAddressPools**.

Field	Type	Description
spec.nodeSelectors	**string**	Optional: **NodeSelectors** limits the nodes to announce as next hops for the load balancer IP. When empty, all the nodes are announced as next hops. **IMPORTANT** Limiting the nodes to announce as next hops is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process. For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope
spec.interfaces	**string**	Optional: The list of **interfaces** that are used to announce the load balancer IP.

35.5.6. Configuring MetalLB with an L2 advertisement

Configure MetalLB as follows so that the **IPAddressPool** is advertised with the L2 protocol.

Prerequisites

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

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

Procedure

1. Create an IP address pool.

   a. Create a file, such as **ipaddresspool.yaml**, with content like the following example:

   ```
   apiVersion: metallb.io/v1beta1
   kind: IPAddressPool
   metadata:
     namespace: metallb-system
     name: doc-example-l2
   spec:
     addresses:
       - 4.4.4.0/24
     autoAssign: false
   ```

   b. Apply the configuration for the IP address pool:

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

2. Create a L2 advertisement.

   a. Create a file, such as **l2advertisement.yaml**, with content like the following example:

   ```
   apiVersion: metallb.io/v1beta1
   kind: L2Advertisement
   metadata:
     name: l2advertisement
     namespace: metallb-system
   spec:
     ipAddressPools:
      - doc-example-l2
   ```

   b. Apply the configuration:

   ```
   $ oc apply -f l2advertisement.yaml
   ```

35.5.7. Configuring MetalLB with a L2 advertisement and label

The **ipAddressPoolSelectors** field in the **BGPAdvertisement** and **L2Advertisement** custom resource definitions is used to associate the **IPAddressPool** to the advertisement based on the label assigned to the **IPAddressPool** instead of the name itself.

This example shows how to configure MetalLB so that the **IPAddressPool** is advertised with the L2 protocol by configuring the **ipAddressPoolSelectors** field.

Prerequisites

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

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

Procedure

1. Create an IP address pool.

   a. Create a file, such as **ipaddresspool.yaml**, with content like the following example:

   ```
   apiVersion: metallb.io/v1beta1
   kind: IPAddressPool
   metadata:
     namespace: metallb-system
     name: doc-example-l2-label
     labels:
       zone: east
   spec:
     addresses:
       - 172.31.249.87/32
   ```

   b. Apply the configuration for the IP address pool:

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

2. Create a L2 advertisement advertising the IP using **ipAddressPoolSelectors**.

a. Create a file, such as **l2advertisement.yaml**, with content like the following example:

```
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: l2advertisement-label
  namespace: metallb-system
spec:
  ipAddressPoolSelectors:
    - matchExpressions:
        - key: zone
          operator: In
          values:
            - east
```

b. Apply the configuration:

```
$ oc apply -f l2advertisement.yaml
```

35.5.8. Configuring MetalLB with an L2 advertisement for selected interfaces

By default, the IP addresses from IP address pool that has been assigned to the service, is advertised from all the network interfaces. The **interfaces** field in the **L2Advertisement** custom resource definition is used to restrict those network interfaces that advertise the IP address pool.

This example shows how to configure MetalLB so that the IP address pool is advertised only from the network interfaces listed in the **interfaces** field of all nodes.

Prerequisites

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

- You are logged in as a user with **cluster-admin** privileges.

Procedure

1. Create an IP address pool.

a. Create a file, such as **ipaddresspool.yaml**, and enter the configuration details like the following example:

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  namespace: metallb-system
  name: doc-example-l2
spec:
  addresses:
    - 4.4.4.0/24
  autoAssign: false
```

b. Apply the configuration for the IP address pool like the following example:

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

2. Create a L2 advertisement advertising the IP with **interfaces** selector.

   a. Create a YAML file, such as **l2advertisement.yaml**, and enter the configuration details like the following example:

   ```
   apiVersion: metallb.io/v1beta1
   kind: L2Advertisement
   metadata:
     name: l2advertisement
     namespace: metallb-system
   spec:
     ipAddressPools:
      - doc-example-l2
     interfaces:
      - interfaceA
      - interfaceB
   ```

   b. Apply the configuration for the advertisement like the following example:

   ```
   $ oc apply -f l2advertisement.yaml
   ```

> **IMPORTANT**
>
> The interface selector does not affect how MetalLB chooses the node to announce a given IP by using L2. The chosen node does not announce the service if the node does not have the selected interface.

35.5.9. Additional resources

- Configuring a community alias.

35.6. CONFIGURING METALLB BGP PEERS

As a cluster administrator, you can add, modify, and delete Border Gateway Protocol (BGP) peers. The MetalLB Operator uses the BGP peer custom resources to identify which peers that MetalLB **speaker** pods contact to start BGP sessions. The peers receive the route advertisements for the load-balancer IP addresses that MetalLB assigns to services.

35.6.1. About the BGP peer custom resource

The fields for the BGP peer custom resource are described in the following table.

Table 35.5. MetalLB BGP peer custom resource

Field	Type	Description
metadata.name	**string**	Specifies the name for the BGP peer custom resource.
metadata.name space	**string**	Specifies the namespace for the BGP peer custom resource.

Field	Type	Description
spec.myASN	**integer**	Specifies the Autonomous System number for the local end of the BGP session. Specify the same value in all BGP peer custom resources that you add. The range is **0** to **4294967295**.
spec.peerASN	**integer**	Specifies the Autonomous System number for the remote end of the BGP session. The range is **0** to **4294967295**.
spec.peerAddress	**string**	Specifies the IP address of the peer to contact for establishing the BGP session.
spec.sourceAddress	**string**	Optional: Specifies the IP address to use when establishing the BGP session. The value must be an IPv4 address.
spec.peerPort	**integer**	Optional: Specifies the network port of the peer to contact for establishing the BGP session. The range is **0** to **16384**.
spec.holdTime	**string**	Optional: Specifies the duration for the hold time to propose to the BGP peer. The minimum value is 3 seconds (**3s**). The common units are seconds and minutes, such as **3s**, **1m**, and **5m30s**. To detect path failures more quickly, also configure BFD.
spec.keepaliveTime	**string**	Optional: Specifies the maximum interval between sending keep-alive messages to the BGP peer. If you specify this field, you must also specify a value for the **holdTime** field. The specified value must be less than the value for the **holdTime** field.
spec.routerID	**string**	Optional: Specifies the router ID to advertise to the BGP peer. If you specify this field, you must specify the same value in every BGP peer custom resource that you add.
spec.password	**string**	Optional: Specifies the MD5 password to send to the peer for routers that enforce TCP MD5 authenticated BGP sessions.
spec.passwordSecret	**string**	Optional: Specifies name of the authentication secret for the BGP Peer. The secret must live in the **metallb** namespace and be of type basic-auth.
spec.bfdProfile	**string**	Optional: Specifies the name of a BFD profile.
spec.nodeSelectors	**object[]**	Optional: Specifies a selector, using match expressions and match labels, to control which nodes can connect to the BGP peer.

Field	Type	Description
spec.ebgpMulti Hop	**boolean**	Optional: Specifies that the BGP peer is multiple network hops away. If the BGP peer is not directly connected to the same network, the speaker cannot establish a BGP session unless this field is set to **true**. This field applies to *external BGP*. External BGP is the term that is used to describe when a BGP peer belongs to a different Autonomous System.

> **NOTE**
>
> The **passwordSecret** field is mutually exclusive with the **password** field, and contains a reference to a secret containing the password to use. Setting both fields results in a failure of the parsing.

35.6.2. Configuring a BGP peer

As a cluster administrator, you can add a BGP peer custom resource to exchange routing information with network routers and advertise the IP addresses for services.

Prerequisites

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

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

- Configure MetalLB with a BGP advertisement.

Procedure

1. Create a file, such as **bgppeer.yaml**, with content like the following example:

   ```
   apiVersion: metallb.io/v1beta2
   kind: BGPPeer
   metadata:
     namespace: metallb-system
     name: doc-example-peer
   spec:
     peerAddress: 10.0.0.1
     peerASN: 64501
     myASN: 64500
     routerID: 10.10.10.10
   ```

2. Apply the configuration for the BGP peer:

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

35.6.3. Configure a specific set of BGP peers for a given address pool

This procedure illustrates how to:

- Configure a set of address pools (**pool1** and **pool2**).

- Configure a set of BGP peers (**peer1** and **peer2**).

- Configure BGP advertisement to assign **pool1** to **peer1** and **pool2** to **peer2**.

Prerequisites

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

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

Procedure

1. Create address pool **pool1**.

   a. Create a file, such as **ipaddresspool1.yaml**, with content like the following example:

   ```
   apiVersion: metallb.io/v1beta1
   kind: IPAddressPool
   metadata:
     namespace: metallb-system
     name: pool1
   spec:
     addresses:
       - 4.4.4.100-4.4.4.200
       - 2001:100:4::200-2001:100:4::400
   ```

   b. Apply the configuration for the IP address pool **pool1**:

   ```
   $ oc apply -f ipaddresspool1.yaml
   ```

2. Create address pool **pool2**.

   a. Create a file, such as **ipaddresspool2.yaml**, with content like the following example:

   ```
   apiVersion: metallb.io/v1beta1
   kind: IPAddressPool
   metadata:
     namespace: metallb-system
     name: pool2
   spec:
     addresses:
       - 5.5.5.100-5.5.5.200
       - 2001:100:5::200-2001:100:5::400
   ```

   b. Apply the configuration for the IP address pool **pool2**:

   ```
   $ oc apply -f ipaddresspool2.yaml
   ```

3. Create BGP **peer1**.

   a. Create a file, such as **bgppeer1.yaml**, with content like the following example:

   ```
   apiVersion: metallb.io/v1beta2
   kind: BGPPeer
   metadata:
   ```

```
  namespace: metallb-system
  name: peer1
spec:
  peerAddress: 10.0.0.1
  peerASN: 64501
  myASN: 64500
  routerID: 10.10.10.10
```

b. Apply the configuration for the BGP peer:

```
$ oc apply -f bgppeer1.yaml
```

4. Create BGP **peer2**.

a. Create a file, such as **bgppeer2.yaml**, with content like the following example:

```
apiVersion: metallb.io/v1beta2
kind: BGPPeer
metadata:
  namespace: metallb-system
  name: peer2
spec:
  peerAddress: 10.0.0.2
  peerASN: 64501
  myASN: 64500
  routerID: 10.10.10.10
```

b. Apply the configuration for the BGP peer2:

```
$ oc apply -f bgppeer2.yaml
```

5. Create BGP advertisement 1.

a. Create a file, such as **bgpadvertisement1.yaml**, with content like the following example:

```
apiVersion: metallb.io/v1beta1
kind: BGPAdvertisement
metadata:
  name: bgpadvertisement-1
  namespace: metallb-system
spec:
  ipAddressPools:
    - pool1
  peers:
    - peer1
  communities:
    - 65535:65282
  aggregationLength: 32
  aggregationLengthV6: 128
  localPref: 100
```

b. Apply the configuration:

```
$ oc apply -f bgpadvertisement1.yaml
```

6. Create BGP advertisement 2.

   a. Create a file, such as **bgpadvertisement2.yaml**, with content like the following example:

   ```
   apiVersion: metallb.io/v1beta1
   kind: BGPAdvertisement
   metadata:
     name: bgpadvertisement-2
     namespace: metallb-system
   spec:
     ipAddressPools:
       - pool2
     peers:
       - peer2
     communities:
       - 65535:65282
     aggregationLength: 32
     aggregationLengthV6: 128
     localPref: 100
   ```

   b. Apply the configuration:

   ```
   $ oc apply -f bgpadvertisement2.yaml
   ```

35.6.4. Example BGP peer configurations

35.6.4.1. Example: Limit which nodes connect to a BGP peer

You can specify the node selectors field to control which nodes can connect to a BGP peer.

```
apiVersion: metallb.io/v1beta2
kind: BGPPeer
metadata:
  name: doc-example-nodesel
  namespace: metallb-system
spec:
  peerAddress: 10.0.20.1
  peerASN: 64501
  myASN: 64500
  nodeSelectors:
  - matchExpressions:
    - key: kubernetes.io/hostname
      operator: In
      values: [compute-1.example.com, compute-2.example.com]
```

35.6.4.2. Example: Specify a BFD profile for a BGP peer

You can specify a BFD profile to associate with BGP peers. BFD compliments BGP by providing more rapid detection of communication failures between peers than BGP alone.

```
apiVersion: metallb.io/v1beta2
kind: BGPPeer
metadata:
```

```
    name: doc-example-peer-bfd
    namespace: metallb-system
  spec:
    peerAddress: 10.0.20.1
    peerASN: 64501
    myASN: 64500
    holdTime: "10s"
    bfdProfile: doc-example-bfd-profile-full
```
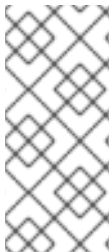
> **NOTE**
>
> Deleting the bidirectional forwarding detection (BFD) profile and removing the **bfdProfile** added to the border gateway protocol (BGP) peer resource does not disable the BFD. Instead, the BGP peer starts using the default BFD profile. To disable BFD from a BGP peer resource, delete the BGP peer configuration and recreate it without a BFD profile. For more information, see BZ#2050824.

35.6.4.3. Example: Specify BGP peers for dual-stack networking

To support dual-stack networking, add one BGP peer custom resource for IPv4 and one BGP peer custom resource for IPv6.

```
apiVersion: metallb.io/v1beta2
kind: BGPPeer
metadata:
  name: doc-example-dual-stack-ipv4
  namespace: metallb-system
spec:
  peerAddress: 10.0.20.1
  peerASN: 64500
  myASN: 64500
---
apiVersion: metallb.io/v1beta2
kind: BGPPeer
metadata:
  name: doc-example-dual-stack-ipv6
  namespace: metallb-system
spec:
  peerAddress: 2620:52:0:88::104
  peerASN: 64500
  myASN: 64500
```

35.6.5. Next steps

- Configuring services to use MetalLB

35.7. CONFIGURING COMMUNITY ALIAS

As a cluster administrator, you can configure a community alias and use it across different advertisements.

35.7.1. About the community custom resource

The **community** custom resource is a collection of aliases for communities. Users can define named aliases to be used when advertising **ipAddressPools** using the **BGPAdvertisement**. The fields for the **community** custom resource are described in the following table.

> **NOTE**
>
> The **community** CRD applies only to BGPAdvertisement.

Table 35.6. MetalLB community custom resource

Field	Type	Description
metadata.name	**string**	Specifies the name for the **community**.
metadata.name space	**string**	Specifies the namespace for the **community**. Specify the same namespace that the MetalLB Operator uses.
spec.communiti es	**string**	Specifies a list of BGP community aliases that can be used in BGPAdvertisements. A community alias consists of a pair of name (alias) and value (number:number). Link the BGPAdvertisement to a community alias by referring to the alias name in its **spec.communities** field.

Table 35.7. CommunityAlias

Field	Type	Description
name	**string**	The name of the alias for the **community**.
value	**string**	The BGP **community** value corresponding to the given name.

35.7.2. Configuring MetalLB with a BGP advertisement and community alias

Configure MetalLB as follows so that the **IPAddressPool** is advertised with the BGP protocol and the community alias set to the numeric value of the NO_ADVERTISE community.

In the following example, the peer BGP router **doc-example-peer-community** receives one **203.0.113.200/32** route and one **fc00:f853:ccd:e799::1/128** route for each load-balancer IP address that MetalLB assigns to a service. A community alias is configured with the **NO_ADVERTISE** community.

Prerequisites

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

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

Procedure

1. Create an IP address pool.

a. Create a file, such as **ipaddresspool.yaml**, with content like the following example:

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  namespace: metallb-system
  name: doc-example-bgp-community
spec:
  addresses:
    - 203.0.113.200/30
    - fc00:f853:ccd:e799::/124
```

b. Apply the configuration for the IP address pool:

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

2. Create a community alias named **community1**.

```
apiVersion: metallb.io/v1beta1
kind: Community
metadata:
  name: community1
  namespace: metallb-system
spec:
  communities:
    - name: NO_ADVERTISE
      value: '65535:65282'
```

3. Create a BGP peer named **doc-example-bgp-peer**.

   a. Create a file, such as **bgppeer.yaml**, with content like the following example:

```
apiVersion: metallb.io/v1beta2
kind: BGPPeer
metadata:
  namespace: metallb-system
  name: doc-example-bgp-peer
spec:
  peerAddress: 10.0.0.1
  peerASN: 64501
  myASN: 64500
  routerID: 10.10.10.10
```

   b. Apply the configuration for the BGP peer:

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

4. Create a BGP advertisement with the community alias.

   a. Create a file, such as **bgpadvertisement.yaml**, with content like the following example:

```
apiVersion: metallb.io/v1beta1
kind: BGPAdvertisement
metadata:
```

```
    name: bgp-community-sample
    namespace: metallb-system
spec:
  aggregationLength: 32
  aggregationLengthV6: 128
  communities:
    - NO_ADVERTISE 1
  ipAddressPools:
    - doc-example-bgp-community
  peers:
    - doc-example-peer
```

[1] Specify the **CommunityAlias.name** here and not the community custom resource (CR) name.

b. Apply the configuration:

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

35.8. CONFIGURING METALLB BFD PROFILES

As a cluster administrator, you can add, modify, and delete Bidirectional Forwarding Detection (BFD) profiles. The MetalLB Operator uses the BFD profile custom resources to identify which BGP sessions use BFD to provide faster path failure detection than BGP alone provides.

35.8.1. About the BFD profile custom resource

The fields for the BFD profile custom resource are described in the following table.

Table 35.8. BFD profile custom resource

Field	Type	Description
metadata.name	**string**	Specifies the name for the BFD profile custom resource.
metadata.name space	**string**	Specifies the namespace for the BFD profile custom resource.
spec.detectMult iplier	**integer**	Specifies the detection multiplier to determine packet loss. The remote transmission interval is multiplied by this value to determine the connection loss detection timer.

For example, when the local system has the detect multiplier set to **3** and the remote system has the transmission interval set to **300**, the local system detects failures only after **900** ms without receiving packets.

The range is **2** to **255**. The default value is **3**. |

Field	Type	Description
spec.echoMode	**boolean**	Specifies the echo transmission mode. If you are not using distributed BFD, echo transmission mode works only when the peer is also FRR. The default value is **false** and echo transmission mode is disabled. When echo transmission mode is enabled, consider increasing the transmission interval of control packets to reduce bandwidth usage. For example, consider increasing the transmit interval to **2000** ms.
spec.echoInterval	**integer**	Specifies the minimum transmission interval, less jitter, that this system uses to send and receive echo packets. The range is **10** to **60000**. The default value is **50** ms.
spec.minimumTtl	**integer**	Specifies the minimum expected TTL for an incoming control packet. This field applies to multi-hop sessions only. The purpose of setting a minimum TTL is to make the packet validation requirements more stringent and avoid receiving control packets from other sessions. The default value is **254** and indicates that the system expects only one hop between this system and the peer.
spec.passiveMode	**boolean**	Specifies whether a session is marked as active or passive. A passive session does not attempt to start the connection. Instead, a passive session waits for control packets from a peer before it begins to reply. Marking a session as passive is useful when you have a router that acts as the central node of a star network and you want to avoid sending control packets that you do not need the system to send. The default value is **false** and marks the session as active.
spec.receiveInterval	**integer**	Specifies the minimum interval that this system is capable of receiving control packets. The range is **10** to **60000**. The default value is **300** ms.
spec.transmitInterval	**integer**	Specifies the minimum transmission interval, less jitter, that this system uses to send control packets. The range is **10** to **60000**. The default value is **300** ms.

35.8.2. Configuring a BFD profile

As a cluster administrator, you can add a BFD profile and configure a BGP peer to use the profile. BFD provides faster path failure detection than BGP alone.

Prerequisites

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

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

Procedure

1. Create a file, such as **bfdprofile.yaml**, with content like the following example:

```
apiVersion: metallb.io/v1beta1
kind: BFDProfile
metadata:
  name: doc-example-bfd-profile-full
  namespace: metallb-system
spec:
  receiveInterval: 300
  transmitInterval: 300
  detectMultiplier: 3
  echoMode: false
  passiveMode: true
  minimumTtl: 254
```

2. Apply the configuration for the BFD profile:

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

35.8.3. Next steps

- Configure a BGP peer to use the BFD profile.

35.9. CONFIGURING SERVICES TO USE METALLB

As a cluster administrator, when you add a service of type **LoadBalancer**, you can control how MetalLB assigns an IP address.

35.9.1. Request a specific IP address

Like some other load–balancer implementations, MetalLB accepts the **spec.loadBalancerIP** field in the service specification.

If the requested IP address is within a range from any address pool, MetalLB assigns the requested IP address. If the requested IP address is not within any range, MetalLB reports a warning.

Example service YAML for a specific IP address

```
apiVersion: v1
kind: Service
metadata:
  name: <service_name>
  annotations:
    metallb.universe.tf/address-pool: <address_pool_name>
spec:
  selector:
    <label_key>: <label_value>
  ports:
```

```
    - port: 8080
      targetPort: 8080
      protocol: TCP
  type: LoadBalancer
  loadBalancerIP: <ip_address>
```

If MetalLB cannot assign the requested IP address, the **EXTERNAL-IP** for the service reports **<pending>** and running **oc describe service <service_name>** includes an event like the following example.

Example event when MetalLB cannot assign a requested IP address

```
  ...
Events:
  Type     Reason          Age    From            Message
  ----     ------          ----   ----            -------
  Warning  AllocationFailed  3m16s  metallb-controller  Failed to allocate IP for "default/invalid-request": "4.3.2.1" is not allowed in config
```

35.9.2. Request an IP address from a specific pool

To assign an IP address from a specific range, but you are not concerned with the specific IP address, then you can use the **metallb.universe.tf/address-pool** annotation to request an IP address from the specified address pool.

Example service YAML for an IP address from a specific pool

```
apiVersion: v1
kind: Service
metadata:
  name: <service_name>
  annotations:
    metallb.universe.tf/address-pool: <address_pool_name>
spec:
  selector:
    <label_key>: <label_value>
  ports:
    - port: 8080
      targetPort: 8080
      protocol: TCP
  type: LoadBalancer
```

If the address pool that you specify for **<address_pool_name>** does not exist, MetalLB attempts to assign an IP address from any pool that permits automatic assignment.

35.9.3. Accept any IP address

By default, address pools are configured to permit automatic assignment. MetalLB assigns an IP address from these address pools.

To accept any IP address from any pool that is configured for automatic assignment, no special annotation or configuration is required.

Example service YAML for accepting any IP address

```
apiVersion: v1
kind: Service
metadata:
  name: <service_name>
spec:
  selector:
    <label_key>: <label_value>
  ports:
    - port: 8080
      targetPort: 8080
      protocol: TCP
  type: LoadBalancer
```

35.9.4. Share a specific IP address

By default, services do not share IP addresses. However, if you need to colocate services on a single IP address, you can enable selective IP sharing by adding the **metallb.universe.tf/allow-shared-ip** annotation to the services.

```
apiVersion: v1
kind: Service
metadata:
  name: service-http
  annotations:
    metallb.universe.tf/address-pool: doc-example
    metallb.universe.tf/allow-shared-ip: "web-server-svc"   1
spec:
  ports:
    - name: http
      port: 80   2
      protocol: TCP
      targetPort: 8080
  selector:
    <label_key>: <label_value>   3
  type: LoadBalancer
  loadBalancerIP: 172.31.249.7   4
---
apiVersion: v1
kind: Service
metadata:
  name: service-https
  annotations:
    metallb.universe.tf/address-pool: doc-example
    metallb.universe.tf/allow-shared-ip: "web-server-svc"   5
spec:
  ports:
    - name: https
      port: 443   6
      protocol: TCP
      targetPort: 8080
  selector:
    <label_key>: <label_value>   7
  type: LoadBalancer
  loadBalancerIP: 172.31.249.7   8
```

1 **5** Specify the same value for the **metallb.universe.tf/allow-shared-ip** annotation. This value is referred to as the *sharing key*.

2 **6** Specify different port numbers for the services.

3 **7** Specify identical pod selectors if you must specify **externalTrafficPolicy: local** so the services send traffic to the same set of pods. If you use the **cluster** external traffic policy, then the pod selectors do not need to be identical.

4 **8** Optional: If you specify the three preceding items, MetalLB might colocate the services on the same IP address. To ensure that services share an IP address, specify the IP address to share.

By default, Kubernetes does not allow multiprotocol load balancer services. This limitation would normally make it impossible to run a service like DNS that needs to listen on both TCP and UDP. To work around this limitation of Kubernetes with MetalLB, create two services:

- For one service, specify TCP and for the second service, specify UDP.

- In both services, specify the same pod selector.

- Specify the same sharing key and **spec.loadBalancerIP** value to colocate the TCP and UDP services on the same IP address.

35.9.5. Configuring a service with MetalLB

You can configure a load-balancing service to use an external IP address from an address pool.

Prerequisites

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

- Install the MetalLB Operator and start MetalLB.

- Configure at least one address pool.

- Configure your network to route traffic from the clients to the host network for the cluster.

Procedure

1. Create a **<service_name>.yaml** file. In the file, ensure that the **spec.type** field is set to **LoadBalancer**.
   Refer to the examples for information about how to request the external IP address that MetalLB assigns to the service.

2. Create the service:

   ```
   $ oc apply -f <service_name>.yaml
   ```

   Example output

   ```
   service/<service_name> created
   ```

Verification

- Describe the service:

  ```
  $ oc describe service <service_name>
  ```

Example output

```
Name:                  <service_name>
Namespace:              default
Labels:                <none>
Annotations:            metallb.universe.tf/address-pool: doc-example  <.>
Selector:              app=service_name
Type:              LoadBalancer  <.>
IP Family Policy:        SingleStack
IP Families:          IPv4
IP:            10.105.237.254
IPs:            10.105.237.254
LoadBalancer Ingress:    192.168.100.5  <.>
Port:                <unset>  80/TCP
TargetPort:            8080/TCP
NodePort:              <unset>  30550/TCP
Endpoints:            10.244.0.50:8080
Session Affinity:        None
External Traffic Policy:  Cluster
Events:  <.>
  Type    Reason        Age            From          Message
  ----    ------        ----            ----          -------
  Normal  nodeAssigned  32m (x2 over 32m)  metallb-speaker  announcing from node "
<node_name>"
```

<.> The annotation is present if you request an IP address from a specific pool. <.> The service type must indicate **LoadBalancer**. <.> The load-balancer ingress field indicates the external IP address if the service is assigned correctly. <.> The events field indicates the node name that is assigned to announce the external IP address. If you experience an error, the events field indicates the reason for the error.

35.10. METALLB LOGGING, TROUBLESHOOTING, AND SUPPORT

If you need to troubleshoot MetalLB configuration, see the following sections for commonly used commands.

35.10.1. Setting the MetalLB logging levels

MetalLB uses FRRouting (FRR) in a container with the default setting of **info** generates a lot of logging. You can control the verbosity of the logs generated by setting the **logLevel** as illustrated in this example.

Gain a deeper insight into MetalLB by setting the **logLevel** to **debug** as follows:

Prerequisites

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

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

Procedure

1. Create a file, such as **setdebugloglevel.yaml**, with content like the following example:

```
apiVersion: metallb.io/v1beta1
kind: MetalLB
metadata:
  name: metallb
  namespace: metallb-system
spec:
  logLevel: debug
  nodeSelector:
    node-role.kubernetes.io/worker: ""
```

2. Apply the configuration:

```
$ oc replace -f setdebugloglevel.yaml
```

> **NOTE**
>
> Use **oc replace** as the understanding is the **metallb** CR is already created and here you are changing the log level.

3. Display the names of the **speaker** pods:

```
$ oc get -n metallb-system pods -l component=speaker
```

Example output

```
NAME             READY   STATUS    RESTARTS   AGE
speaker-2m9pm     4/4    Running   0          9m19s
speaker-7m4qw     3/4    Running   0          19s
speaker-szlmx     4/4    Running   0          9m19s
```

> **NOTE**
>
> Speaker and controller pods are recreated to ensure the updated logging level is applied. The logging level is modified for all the components of MetalLB.

4. View the **speaker** logs:

```
$ oc logs -n metallb-system speaker-7m4qw -c speaker
```

Example output

```
{"branch":"main","caller":"main.go:92","commit":"3d052535","goversion":"gc / go1.17.1 /
amd64","level":"info","msg":"MetalLB speaker starting (commit 3d052535, branch
main)","ts":"2022-05-17T09:55:05Z","version":""}
{"caller":"announcer.go:110","event":"createARPResponder","interface":"ens4","level":"info","m
sg":"created ARP responder for interface","ts":"2022-05-17T09:55:05Z"}
{"caller":"announcer.go:119","event":"createNDPResponder","interface":"ens4","level":"info","m
sg":"created NDP responder for interface","ts":"2022-05-17T09:55:05Z"}
```

{"caller":"announcer.go:110","event":"createARPResponder","interface":"tun0","level":"info","msg":"created ARP responder for interface","ts":"2022-05-17T09:55:05Z"}
{"caller":"announcer.go:119","event":"createNDPResponder","interface":"tun0","level":"info","msg":"created NDP responder for interface","ts":"2022-05-17T09:55:05Z"}
I0517 09:55:06.515686   95 request.go:665] Waited for 1.026500832s due to client-side throttling, not priority and fairness, request: GET:https://172.30.0.1:443/apis/operators.coreos.com/v1alpha1?timeout=32s
{"Starting Manager":"(MISSING)","caller":"k8s.go:389","level":"info","ts":"2022-05-17T09:55:08Z"}
{"caller":"speakerlist.go:310","level":"info","msg":"node event - forcing sync","node addr":"10.0.128.4","node event":"NodeJoin","node name":"ci-ln-qb8t3mb-72292-7s7rh-worker-a-vvznj","ts":"2022-05-17T09:55:08Z"}
{"caller":"service_controller.go:113","controller":"ServiceReconciler","enqueueing":"openshift-kube-controller-manager-operator/metrics","epslice":"{\"metadata\":{\"name\":\"metrics-xtsxr\",\"generateName\":\"metrics-\",\"namespace\":\"openshift-kube-controller-manager-operator\",\"uid\":\"ac6766d7-8504-492c-9d1e-4ae8897990ad\",\"resourceVersion\":\"9041\",\"generation\":4,\"creationTimestamp\":\"2022-05-17T07:16:53Z\",\"labels\":{\"app\":\"kube-controller-manager-operator\",\"endpointslice.kubernetes.io/managed-by\":\"endpointslice-controller.k8s.io\",\"kubernetes.io/service-name\":\"metrics\"},\"annotations\":{\"endpoints.kubernetes.io/last-change-trigger-time\":\"2022-05-17T07:21:34Z\"},\"ownerReferences\":[{\"apiVersion\":\"v1\",\"kind\":\"Service\",\"name\":\"metrics\",\"uid\":\"0518eed3-6152-42be-b566-0bd00a60faf8\",\"controller\":true,\"blockOwnerDeletion\":true}],\"managedFields\":[{\"manager\":\"kube-controller-manager\",\"operation\":\"Update\",\"apiVersion\":\"discovery.k8s.io/v1\",\"time\":\"2022-05-17T07:20:02Z\",\"fieldsType\":\"FieldsV1\",\"fieldsV1\":{\"f:addressType\":{},\"f:endpoints\":{},\"f:metadata\":{\"f:annotations\":{\".\":{},\"f:endpoints.kubernetes.io/last-change-trigger-time\":{}},\"f:generateName\":{},\"f:labels\":{\".\":{},\"f:app\":{},\"f:endpointslice.kubernetes.io/managed-by\":{},\"f:kubernetes.io/service-name\":{}},\"f:ownerReferences\":{\".\":{},\"k:{\\\"uid\\\":\\\"0518eed3-6152-42be-b566-0bd00a60faf8\\\"}\":{}}},\"f:ports\":{}}}]},\"addressType\":\"IPv4\",\"endpoints\":[{\"addresses\":[\"10.129.0.7\"],\"conditions\":{\"ready\":true,\"serving\":true,\"terminating\":false},\"targetRef\":{\"kind\":\"Pod\",\"namespace\":\"openshift-kube-controller-manager-operator\",\"name\":\"kube-controller-manager-operator-6b98b89ddd-8d4nf\",\"uid\":\"dd5139b8-e41c-4946-a31b-1a629314e844\",\"resourceVersion\":\"9038\"},\"nodeName\":\"ci-ln-qb8t3mb-72292-7s7rh-master-0\",\"zone\":\"us-central1-a\"}],\"ports\":[{\"name\":\"https\",\"protocol\":\"TCP\",\"port\":8443}]}","level":"debug","ts":"2022-05-17T09:55:08Z"}

5. View the FRR logs:

```
$ oc logs -n metallb-system speaker-7m4qw -c frr
```

Example output

```
Started watchfrr
2022/05/17 09:55:05 ZEBRA: client 16 says hello and bids fair to announce only bgp routes vrf=0
2022/05/17 09:55:05 ZEBRA: client 31 says hello and bids fair to announce only vnc routes vrf=0
2022/05/17 09:55:05 ZEBRA: client 38 says hello and bids fair to announce only static routes vrf=0
2022/05/17 09:55:05 ZEBRA: client 43 says hello and bids fair to announce only bfd routes
```

vrf=0
2022/05/17 09:57:25.089 BGP: Creating Default VRF, AS 64500
2022/05/17 09:57:25.090 BGP: dup addr detect enable max_moves 5 time 180 freeze
disable freeze_time 0
2022/05/17 09:57:25.090 BGP: bgp_get: Registering BGP instance (null) to zebra
2022/05/17 09:57:25.090 BGP: Registering VRF 0
2022/05/17 09:57:25.091 BGP: Rx Router Id update VRF 0 Id 10.131.0.1/32
2022/05/17 09:57:25.091 BGP: RID change : vrf VRF default(0), RTR ID 10.131.0.1
2022/05/17 09:57:25.091 BGP: Rx Intf add VRF 0 IF br0
2022/05/17 09:57:25.091 BGP: Rx Intf add VRF 0 IF ens4
2022/05/17 09:57:25.091 BGP: Rx Intf address add VRF 0 IF ens4 addr 10.0.128.4/32
2022/05/17 09:57:25.091 BGP: Rx Intf address add VRF 0 IF ens4 addr
fe80::c9d:84da:4d86:5618/64
2022/05/17 09:57:25.091 BGP: Rx Intf add VRF 0 IF lo
2022/05/17 09:57:25.091 BGP: Rx Intf add VRF 0 IF ovs-system
2022/05/17 09:57:25.091 BGP: Rx Intf add VRF 0 IF tun0
2022/05/17 09:57:25.091 BGP: Rx Intf address add VRF 0 IF tun0 addr 10.131.0.1/23
2022/05/17 09:57:25.091 BGP: Rx Intf address add VRF 0 IF tun0 addr
fe80::40f1:d1ff:feb6:5322/64
2022/05/17 09:57:25.091 BGP: Rx Intf add VRF 0 IF veth2da49fed
2022/05/17 09:57:25.091 BGP: Rx Intf address add VRF 0 IF veth2da49fed addr
fe80::24bd:d1ff:fec1:d88/64
2022/05/17 09:57:25.091 BGP: Rx Intf add VRF 0 IF veth2fa08c8c
2022/05/17 09:57:25.091 BGP: Rx Intf address add VRF 0 IF veth2fa08c8c addr
fe80::6870:ff:fe96:efc8/64
2022/05/17 09:57:25.091 BGP: Rx Intf add VRF 0 IF veth41e356b7
2022/05/17 09:57:25.091 BGP: Rx Intf address add VRF 0 IF veth41e356b7 addr
fe80::48ff:37ff:fede:eb4b/64
2022/05/17 09:57:25.092 BGP: Rx Intf add VRF 0 IF veth1295c6e2
2022/05/17 09:57:25.092 BGP: Rx Intf address add VRF 0 IF veth1295c6e2 addr
fe80::b827:a2ff:feed:637/64
2022/05/17 09:57:25.092 BGP: Rx Intf add VRF 0 IF veth9733c6dc
2022/05/17 09:57:25.092 BGP: Rx Intf address add VRF 0 IF veth9733c6dc addr
fe80::3cf4:15ff:fe11:e541/64
2022/05/17 09:57:25.092 BGP: Rx Intf add VRF 0 IF veth336680ea
2022/05/17 09:57:25.092 BGP: Rx Intf address add VRF 0 IF veth336680ea addr
fe80::94b1:8bff:fe7e:488c/64
2022/05/17 09:57:25.092 BGP: Rx Intf add VRF 0 IF vetha0a907b7
2022/05/17 09:57:25.092 BGP: Rx Intf address add VRF 0 IF vetha0a907b7 addr
fe80::3855:a6ff:fe73:46c3/64
2022/05/17 09:57:25.092 BGP: Rx Intf add VRF 0 IF vethf35a4398
2022/05/17 09:57:25.092 BGP: Rx Intf address add VRF 0 IF vethf35a4398 addr
fe80::40ef:2fff:fe57:4c4d/64
2022/05/17 09:57:25.092 BGP: Rx Intf add VRF 0 IF vethf831b7f4
2022/05/17 09:57:25.092 BGP: Rx Intf address add VRF 0 IF vethf831b7f4 addr
fe80::f0d9:89ff:fe7c:1d32/64
2022/05/17 09:57:25.092 BGP: Rx Intf add VRF 0 IF vxlan_sys_4789
2022/05/17 09:57:25.092 BGP: Rx Intf address add VRF 0 IF vxlan_sys_4789 addr
fe80::80c1:82ff:fe4b:f078/64
2022/05/17 09:57:26.094 BGP: 10.0.0.1 [FSM] Timer (start timer expire).
2022/05/17 09:57:26.094 BGP: 10.0.0.1 [FSM] BGP_Start (Idle->Connect), fd -1
2022/05/17 09:57:26.094 BGP: Allocated bnc 10.0.0.1/32(0)(VRF default) peer
0x7f807f7631a0
2022/05/17 09:57:26.094 BGP: sendmsg_zebra_rnh: sending cmd
ZEBRA_NEXTHOP_REGISTER for 10.0.0.1/32 (vrf VRF default)
2022/05/17 09:57:26.094 BGP: 10.0.0.1 [FSM] Waiting for NHT

> 2022/05/17 09:57:26.094 BGP: bgp_fsm_change_status : vrf default(0), Status: Connect established_peers 0
> 2022/05/17 09:57:26.094 BGP: 10.0.0.1 went from Idle to Connect
> 2022/05/17 09:57:26.094 BGP: 10.0.0.1 [FSM] TCP_connection_open_failed (Connect->Active), fd -1
> 2022/05/17 09:57:26.094 BGP: bgp_fsm_change_status : vrf default(0), Status: Active established_peers 0
> 2022/05/17 09:57:26.094 BGP: 10.0.0.1 went from Connect to Active
> 2022/05/17 09:57:26.094 ZEBRA: rnh_register msg from client bgp: hdr->length=8, type=nexthop vrf=0
> 2022/05/17 09:57:26.094 ZEBRA: 0: Add RNH 10.0.0.1/32 type Nexthop
> 2022/05/17 09:57:26.094 ZEBRA: 0:10.0.0.1/32: Evaluate RNH, type Nexthop (force)
> 2022/05/17 09:57:26.094 ZEBRA: 0:10.0.0.1/32: NH has become unresolved
> 2022/05/17 09:57:26.094 ZEBRA: 0: Client bgp registers for RNH 10.0.0.1/32 type Nexthop
> 2022/05/17 09:57:26.094 BGP: VRF default(0): Rcvd NH update 10.0.0.1/32(0) - metric 0/0 #nhops 0/0 flags 0x6
> 2022/05/17 09:57:26.094 BGP: NH update for 10.0.0.1/32(0)(VRF default) - flags 0x6 chgflags 0x0 - evaluate paths
> 2022/05/17 09:57:26.094 BGP: evaluate_paths: Updating peer (10.0.0.1(VRF default)) status with NHT
> 2022/05/17 09:57:30.081 ZEBRA: Event driven route-map update triggered
> 2022/05/17 09:57:30.081 ZEBRA: Event handler for route-map: 10.0.0.1-out
> 2022/05/17 09:57:30.081 ZEBRA: Event handler for route-map: 10.0.0.1-in
> 2022/05/17 09:57:31.104 ZEBRA: netlink_parse_info: netlink-listen (NS 0) type RTM_NEWNEIGH(28), len=76, seq=0, pid=0
> 2022/05/17 09:57:31.104 ZEBRA: Neighbor Entry received is not on a VLAN or a BRIDGE, ignoring
> 2022/05/17 09:57:31.105 ZEBRA: netlink_parse_info: netlink-listen (NS 0) type RTM_NEWNEIGH(28), len=76, seq=0, pid=0
> 2022/05/17 09:57:31.105 ZEBRA: Neighbor Entry received is not on a VLAN or a BRIDGE, ignoring

35.10.1.1. FRRouting (FRR) log levels

The following table describes the FRR logging levels.

Table 35.9. Log levels

Log level	Description
all	Supplies all logging information for all logging levels.
debug	Information that is diagnostically helpful to people. Set to **debug** to give detailed troubleshooting information.
info	Provides information that always should be logged but under normal circumstances does not require user intervention. This is the default logging level.
warn	Anything that can potentially cause inconsistent **MetalLB** behaviour. Usually **MetalLB** automatically recovers from this type of error.

Log level	Description
error	Any error that is fatal to the functioning of **MetalLB**. These errors usually require administrator intervention to fix.
none	Turn off all logging.

35.10.2. Troubleshooting BGP issues

The BGP implementation that Red Hat supports uses FRRouting (FRR) in a container in the **speaker** pods. As a cluster administrator, if you need to troubleshoot BGP configuration issues, you need to run commands in the FRR container.

Prerequisites

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

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

Procedure

1. Display the names of the **speaker** pods:

   ```
   $ oc get -n metallb-system pods -l component=speaker
   ```

   Example output

   ```
   NAME          READY   STATUS    RESTARTS   AGE
   speaker-66bth  4/4     Running   0          56m
   speaker-gvfnf  4/4     Running   0          56m
   ...
   ```

2. Display the running configuration for FRR:

   ```
   $ oc exec -n metallb-system speaker-66bth -c frr -- vtysh -c "show running-config"
   ```

   Example output

   ```
   Building configuration...

   Current configuration:
   !
   frr version 7.5.1_git
   frr defaults traditional
   hostname some-hostname
   log file /etc/frr/frr.log informational
   log timestamp precision 3
   service integrated-vtysh-config
   !
   router bgp 64500          1
    bgp router-id 10.0.1.2
   ```

```
   no bgp ebgp-requires-policy
   no bgp default ipv4-unicast
   no bgp network import-check
   neighbor 10.0.2.3 remote-as 64500            2
   neighbor 10.0.2.3 bfd profile doc-example-bfd-profile-full       3
   neighbor 10.0.2.3 timers 5 15
   neighbor 10.0.2.4 remote-as 64500            4
   neighbor 10.0.2.4 bfd profile doc-example-bfd-profile-full       5
   neighbor 10.0.2.4 timers 5 15
   !
   address-family ipv4 unicast
    network 203.0.113.200/30        6
    neighbor 10.0.2.3 activate
    neighbor 10.0.2.3 route-map 10.0.2.3-in in
    neighbor 10.0.2.4 activate
    neighbor 10.0.2.4 route-map 10.0.2.4-in in
   exit-address-family
    !
   address-family ipv6 unicast
    network fc00:f853:ccd:e799::/124         7
    neighbor 10.0.2.3 activate
    neighbor 10.0.2.3 route-map 10.0.2.3-in in
    neighbor 10.0.2.4 activate
    neighbor 10.0.2.4 route-map 10.0.2.4-in in
   exit-address-family
   !
   route-map 10.0.2.3-in deny 20
   !
   route-map 10.0.2.4-in deny 20
   !
   ip nht resolve-via-default
   !
   ipv6 nht resolve-via-default
   !
   line vty
   !
   bfd
    profile doc-example-bfd-profile-full        8
    transmit-interval 35
    receive-interval 35
    passive-mode
    echo-mode
    echo-interval 35
    minimum-ttl 10
   !
   !
   end
```

<.> The **router bgp** section indicates the ASN for MetalLB. <.> Confirm that a **neighbor <ip-address> remote-as <peer-ASN>** line exists for each BGP peer custom resource that you added. <.> If you configured BFD, confirm that the BFD profile is associated with the correct BGP peer and that the BFD profile appears in the command output. <.> Confirm that the **network <ip-address-range>** lines match the IP address ranges that you specified in address pool custom resources that you added.

3. Display the BGP summary:

```
$ oc exec -n metallb-system speaker-66bth -c frr -- vtysh -c "show bgp summary"
```

Example output

```
IPv4 Unicast Summary:
BGP router identifier 10.0.1.2, local AS number 64500 vrf-id 0
BGP table version 1
RIB entries 1, using 192 bytes of memory
Peers 2, using 29 KiB of memory

Neighbor      V        AS  MsgRcvd  MsgSent  TblVer  InQ OutQ  Up/Down State/PfxRcd
PfxSnt
10.0.2.3      4     64500      387      389       0    0    0 00:32:02          0     1  1
10.0.2.4      4     64500        0        0       0    0    0    never     Active     0  2

Total number of neighbors 2

IPv6 Unicast Summary:
BGP router identifier 10.0.1.2, local AS number 64500 vrf-id 0
BGP table version 1
RIB entries 1, using 192 bytes of memory
Peers 2, using 29 KiB of memory

Neighbor      V        AS  MsgRcvd  MsgSent  TblVer  InQ OutQ  Up/Down State/PfxRcd
PfxSnt
10.0.2.3      4     64500      387      389       0    0    0 00:32:02 NoNeg  3
10.0.2.4      4     64500        0        0       0    0    0    never     Active     0  4

Total number of neighbors 2
```

1 **1** **3** Confirm that the output includes a line for each BGP peer custom resource that you added.

2 **4** **2** **4** Output that shows **0** messages received and messages sent indicates a BGP peer that does not have a BGP session. Check network connectivity and the BGP configuration of the BGP peer.

4. Display the BGP peers that received an address pool:

```
$ oc exec -n metallb-system speaker-66bth -c frr -- vtysh -c "show bgp ipv4 unicast
203.0.113.200/30"
```

Replace **ipv4** with **ipv6** to display the BGP peers that received an IPv6 address pool. Replace **203.0.113.200/30** with an IPv4 or IPv6 IP address range from an address pool.

Example output

```
BGP routing table entry for 203.0.113.200/30
Paths: (1 available, best #1, table default)
  Advertised to non peer-group peers:
  10.0.2.3  <.>
```

```
Local
  0.0.0.0 from 0.0.0.0 (10.0.1.2)
    Origin IGP, metric 0, weight 32768, valid, sourced, local, best (First path received)
    Last update: Mon Jan 10 19:49:07 2022
```

<.> Confirm that the output includes an IP address for a BGP peer.

35.10.3. Troubleshooting BFD issues

The Bidirectional Forwarding Detection (BFD) implementation that Red Hat supports uses FRRouting (FRR) in a container in the **speaker** pods. The BFD implementation relies on BFD peers also being configured as BGP peers with an established BGP session. As a cluster administrator, if you need to troubleshoot BFD configuration issues, you need to run commands in the FRR container.

Prerequisites

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

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

Procedure

1. Display the names of the **speaker** pods:

   ```
   $ oc get -n metallb-system pods -l component=speaker
   ```

   Example output

   ```
   NAME          READY  STATUS   RESTARTS  AGE
   speaker-66bth 4/4    Running  0         26m
   speaker-gvfnf 4/4    Running  0         26m
   ...
   ```

2. Display the BFD peers:

   ```
   $ oc exec -n metallb-system speaker-66bth -c frr -- vtysh -c "show bfd peers brief"
   ```

   Example output

   ```
   Session count: 2
   SessionId  LocalAddress        PeerAddress        Status
   =========  ============        ===========        ======
   3909139637 10.0.1.2            10.0.2.3           up <.>
   ```

   <.> Confirm that the **PeerAddress** column includes each BFD peer. If the output does not list a BFD peer IP address that you expected the output to include, troubleshoot BGP connectivity with the peer. If the status field indicates **down**, check for connectivity on the links and equipment between the node and the peer. You can determine the node name for the speaker pod with a command like **oc get pods -n metallb-system speaker-66bth -o jsonpath='{.spec.nodeName}'**.

35.10.4. MetalLB metrics for BGP and BFD

OpenShift Container Platform captures the following metrics that are related to MetalLB and BGP peers and BFD profiles:

- **metallb_bfd_control_packet_input** counts the number of BFD control packets received from each BFD peer.

- **metallb_bfd_control_packet_output** counts the number of BFD control packets sent to each BFD peer.

- **metallb_bfd_echo_packet_input** counts the number of BFD echo packets received from each BFD peer.

- **metallb_bfd_echo_packet_output** counts the number of BFD echo packets sent to each BFD peer.

- **metallb_bfd_session_down_events** counts the number of times the BFD session with a peer entered the **down** state.

- **metallb_bfd_session_up** indicates the connection state with a BFD peer. **1** indicates the session is **up** and **0** indicates the session is **down**.

- **metallb_bfd_session_up_events** counts the number of times the BFD session with a peer entered the **up** state.

- **metallb_bfd_zebra_notifications** counts the number of BFD Zebra notifications for each BFD peer.

- **metallb_bgp_announced_prefixes_total** counts the number of load balancer IP address prefixes that are advertised to BGP peers. The terms *prefix* and *aggregated route* have the same meaning.

- **metallb_bgp_session_up** indicates the connection state with a BGP peer. **1** indicates the session is **up** and **0** indicates the session is **down**.

- **metallb_bgp_updates_total** counts the number of BGP **update** messages that were sent to a BGP peer.

Additional resources

- See Querying metrics for information about using the monitoring dashboard.

35.10.5. About collecting MetalLB data

You can use the **oc adm must-gather** CLI command to collect information about your cluster, your MetalLB configuration, and the MetalLB Operator. The following features and objects are associated with MetalLB and the MetalLB Operator:

- The namespace and child objects that the MetalLB Operator is deployed in

- All MetalLB Operator custom resource definitions (CRDs)

The **oc adm must-gather** CLI command collects the following information from FRRouting (FRR) that Red Hat uses to implement BGP and BFD:

- **/etc/frr/frr.conf**

- **/etc/frr/frr.log**

- **/etc/frr/daemons** configuration file

- **/etc/frr/vtysh.conf**

The log and configuration files in the preceding list are collected from the **frr** container in each **speaker** pod.

In addition to the log and configuration files, the **oc adm must-gather** CLI command collects the output from the following **vtysh** commands:

- **show running-config**

- **show bgp ipv4**

- **show bgp ipv6**

- **show bgp neighbor**

- **show bfd peer**

No additional configuration is required when you run the **oc adm must-gather** CLI command.

Additional resources

- Gathering data about your cluster

CHAPTER 36. ASSOCIATING SECONDARY INTERFACES METRICS TO NETWORK ATTACHMENTS

36.1. EXTENDING SECONDARY NETWORK METRICS FOR MONITORING

Secondary devices, or interfaces, are used for different purposes. It is important to have a way to classify them to be able to aggregate the metrics for secondary devices with the same classification.

Exposed metrics contain the interface but do not specify where the interface originates. This is workable when there are no additional interfaces. However, if secondary interfaces are added, it can be difficult to use the metrics since it is hard to identify interfaces using only interface names.

When adding secondary interfaces, their names depend on the order in which they are added, and different secondary interfaces might belong to different networks and can be used for different purposes.

With **pod_network_name_info** it is possible to extend the current metrics with additional information that identifies the interface type. In this way, it is possible to aggregate the metrics and to add specific alarms to specific interface types.

The network type is generated using the name of the related **NetworkAttachmentDefinition**, that in turn is used to differentiate different classes of secondary networks. For example, different interfaces belonging to different networks or using different CNIs use different network attachment definition names.

36.1.1. Network Metrics Daemon

The Network Metrics Daemon is a daemon component that collects and publishes network related metrics.

The kubelet is already publishing network related metrics you can observe. These metrics are:

- **container_network_receive_bytes_total**

- **container_network_receive_errors_total**

- **container_network_receive_packets_total**

- **container_network_receive_packets_dropped_total**

- **container_network_transmit_bytes_total**

- **container_network_transmit_errors_total**

- **container_network_transmit_packets_total**

- **container_network_transmit_packets_dropped_total**

The labels in these metrics contain, among others:

- Pod name

- Pod namespace

- Interface name (such as **eth0**)

These metrics work well until new interfaces are added to the pod, for example via Multus, as it is not clear what the interface names refer to.

The interface label refers to the interface name, but it is not clear what that interface is meant for. In case of many different interfaces, it would be impossible to understand what network the metrics you are monitoring refer to.

This is addressed by introducing the new **pod_network_name_info** described in the following section.

36.1.2. Metrics with network name

This daemonset publishes a **pod_network_name_info** gauge metric, with a fixed value of **0**:

```
pod_network_name_info{interface="net0",namespace="namespacename",network_name="nadname
space/firstNAD",pod="podname"} 0
```

The network name label is produced using the annotation added by Multus. It is the concatenation of the namespace the network attachment definition belongs to, plus the name of the network attachment definition.

The new metric alone does not provide much value, but combined with the network related **container_network_*** metrics, it offers better support for monitoring secondary networks.

Using a **promql** query like the following ones, it is possible to get a new metric containing the value and the network name retrieved from the **k8s.v1.cni.cncf.io/network-status** annotation:

```
(container_network_receive_bytes_total) + on(namespace,pod,interface) group_left(network_name) (
pod_network_name_info )
(container_network_receive_errors_total) + on(namespace,pod,interface) group_left(network_name) (
pod_network_name_info )
(container_network_receive_packets_total) + on(namespace,pod,interface)
group_left(network_name) ( pod_network_name_info )
(container_network_receive_packets_dropped_total) + on(namespace,pod,interface)
group_left(network_name) ( pod_network_name_info )
(container_network_transmit_bytes_total) + on(namespace,pod,interface) group_left(network_name)
( pod_network_name_info )
(container_network_transmit_errors_total) + on(namespace,pod,interface) group_left(network_name)
( pod_network_name_info )
(container_network_transmit_packets_total) + on(namespace,pod,interface)
group_left(network_name) ( pod_network_name_info )
(container_network_transmit_packets_dropped_total) + on(namespace,pod,interface)
group_left(network_name)
```