Networking

Configuring and managing cluster networking
OpenShift Container Platform 4.4 Networking

Configuring and managing cluster networking
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. UNDERSTANDING NETWORKING

1.1. OPENSHIFT CONTAINER PLATFORM DNS .......................... 7

## CHAPTER 2. ACCESSING HOSTS

2.1. ACCESSING HOSTS ON AMAZON WEB SERVICES IN AN INSTALLER-PROVISIONED INFRASTRUCTURE CLUSTER .......................... 8

## CHAPTER 3. CLUSTER NETWORK OPERATOR IN OPENSHIFT CONTAINER PLATFORM

3.1. CLUSTER NETWORK OPERATOR ........................................ 9
3.2. VIEWING THE CLUSTER NETWORK CONFIGURATION ................. 9
3.3. VIEWING CLUSTER NETWORK OPERATOR STATUS ................... 10
3.4. VIEWING CLUSTER NETWORK OPERATOR LOGS ..................... 10
3.5. CLUSTER NETWORK OPERATOR CONFIGURATION .................. 11
   3.5.1. Configuration parameters for the OpenShift SDN default CNI network provider ............................................. 12
   3.5.2. Configuration parameters for the OVN-Kubernetes default CNI network provider ............................................. 12
   3.5.3. Cluster Network Operator example configuration ............... 13

## CHAPTER 4. DNS OPERATOR IN OPENSHIFT CONTAINER PLATFORM

4.1. DNS OPERATOR ....................................................... 14
4.2. VIEW THE DEFAULT DNS ............................................. 14
4.3. USING DNS FORWARDING .......................................... 15
4.4. DNS OPERATOR STATUS ............................................. 17
4.5. DNS OPERATOR LOGS ............................................... 17

## CHAPTER 5. INGRESS OPERATOR IN OPENSHIFT CONTAINER PLATFORM

5.1. THE INGRESS CONFIGURATION ASSET ................................ 18
5.2. INGRESS CONTROLLER CONFIGURATION PARAMETERS ............ 18
   5.2.1. Ingress controller TLS profiles .................................. 21
   5.2.2. Ingress controller endpoint publishing strategy .................. 23
5.3. VIEW THE DEFAULT INGRESS CONTROLLER ......................... 23
5.4. VIEW INGRESS OPERATOR STATUS .................................. 23
5.5. VIEW INGRESS CONTROLLER LOGS .................................. 24
5.6. VIEW INGRESS CONTROLLER STATUS ................................ 24
5.7. SETTING A CUSTOM DEFAULT CERTIFICATE .......................... 24
5.8. SCALING AN INGRESS CONTROLLER .................................. 26
5.9. CONFIGURING INGRESS CONTROLLER SHARDING BY USING ROUTE LABELS ......................................................... 26
5.10. CONFIGURING INGRESS CONTROLLER SHARDING BY USING NAMESPACE LABELS ..................................................... 27
5.11. CONFIGURING AN INGRESS CONTROLLER TO USE AN INTERNAL LOAD BALANCER ................................................. 27
5.12. CONFIGURING THE DEFAULT INGRESS CONTROLLER FOR YOUR CLUSTER TO BE INTERNAL ............................................. 29
5.13. CONFIGURING THE ROUTE ADMISSION POLICY .................... 30
5.14. ADDITIONAL RESOURCES ............................................. 30

## CHAPTER 6. CONFIGURING NETWORK POLICY WITH OPENSHIFT SDN

6.1. ABOUT NETWORK POLICY ............................................. 31
6.2. EXAMPLE NETWORKPOLICY OBJECT ................................ 33
6.3. CREATING A NETWORKPOLICY OBJECT ............................ 34
6.4. DELETING A NETWORKPOLICY OBJECT ............................. 34
6.5. VIEWING NETWORKPOLICY OBJECTS ................................ 35
6.6. CONFIGURING MULTITENANT ISOLATION USING NETWORKPOLICY ................................................................. 35
6.7. CREATING DEFAULT NETWORK POLICIES FOR A NEW PROJECT ................................................................. 37
   6.7.1. Modifying the template for new projects .......................... 37
   6.7.2. Adding network policy objects to the new project template .................. 38
CHAPTER 7. USING THE STREAM CONTROL TRANSMISSION PROTOCOL (SCTP) ON A BARE METAL CLUSTER

7.1. SUPPORT FOR STREAM CONTROL TRANSMISSION PROTOCOL (SCTP) ON OPENSHIFT CONTAINER PLATFORM
   7.1.1. Example configurations using SCTP protocol

7.2. ENABLING STREAM CONTROL TRANSMISSION PROTOCOL (SCTP)

7.3. VERIFYING STREAM CONTROL TRANSMISSION PROTOCOL (SCTP) IS ENABLED

CHAPTER 8. MULTIPLE NETWORKS

8.1. UNDERSTANDING MULTIPLE NETWORKS
   8.1.1. Usage scenarios for an additional network
   8.1.2. Additional networks in OpenShift Container Platform

8.2. ATTACHING A POD TO AN ADDITIONAL NETWORK
   8.2.1. Adding a Pod to an additional network
      8.2.1.1. Specifying Pod-specific addressing and routing options

8.3. REMOVING A POD FROM AN ADDITIONAL NETWORK
   8.3.1. Removing a Pod from an additional network

8.4. CONFIGURING A BRIDGE NETWORK
   8.4.1. Creating an additional network attachment with the bridge CNI plug-in
      8.4.1.1. Configuration for bridge
      8.4.1.1.1. bridge configuration example
      8.4.1.2. Configuration for ipam CNI plug-in
      8.4.1.2.1. Static IP address assignment configuration example
      8.4.1.2.2. Dynamic IP address assignment configuration example

8.5. CONFIGURING A MACVLAN NETWORK
   8.5.1. Creating an additional network attachment with the macvlan CNI plug-in
      8.5.1.1. Configuration for macvlan CNI plug-in
      8.5.1.1.1. macvlan configuration example
      8.5.1.2. Configuration for ipam CNI plug-in
      8.5.1.2.1. Static ipam configuration YAML
      8.5.1.2.2. Dynamic ipam configuration YAML
      8.5.1.2.3. Static IP address assignment configuration example
      8.5.1.2.4. Dynamic IP address assignment configuration example

8.6. CONFIGURING AN IPVLAN NETWORK
   8.6.1. Creating an additional network attachment with the ipvlan CNI plug-in
      8.6.1.1. Configuration for ipvlan
      8.6.1.1.1. ipvlan configuration example
      8.6.1.2. Configuration for ipam CNI plug-in
      8.6.1.2.1. Static IP address assignment configuration example
      8.6.1.2.2. Dynamic IP address assignment configuration example

8.7. CONFIGURING A HOST-DEVICE NETWORK
   8.7.1. Creating an additional network attachment with the host-device CNI plug-in
      8.7.1.1. Configuration for host-device
      8.7.1.1.1. host-device configuration example
      8.7.1.2. Configuration for ipam CNI plug-in
      8.7.1.2.1. Static IP address assignment configuration example
      8.7.1.2.2. Dynamic IP address assignment configuration example

8.8. EDITING AN ADDITIONAL NETWORK
   8.8.1. Modifying an additional network attachment definition

8.9. REMOVING AN ADDITIONAL NETWORK
   8.9.1. Removing an additional network attachment definition

8.10. CONFIGURING PTP
   8.10.1. About PTP hardware on OpenShift Container Platform
8.10.2. Installing the PTP Operator
8.10.2.1. Installing the Operator using the CLI
8.10.2.2. Installing the Operator using the web console
8.10.3. Automated discovery of PTP network devices
8.10.4. Configuring Linuxptp services

CHAPTER 9. HARDWARE NETWORKS

9.1. ABOUT SINGLE ROOT I/O VIRTUALIZATION (SR-IOV) HARDWARE NETWORKS
9.1.1. About SR-IOV hardware on OpenShift Container Platform
9.1.1.1. Supported devices
9.1.1.2. Example use of a virtual function (VF) in a Pod
9.2. INSTALLING THE SR-IOV NETWORK OPERATOR
9.2.1. Installing SR-IOV Network Operator
9.2.1.1. Installing the Operator using the CLI
9.2.1.2. Installing the Operator using the web console
9.3. CONFIGURING THE SR-IOV NETWORK OPERATOR
9.3.1. Configuring the SR-IOV Network Operator
9.3.1.1. About the Network Resources Injector
9.3.1.2. About the SR-IOV Operator admission controller webhook
9.3.1.3. About custom node selectors
9.3.1.4. Disabling or enabling the Network Resources Injector
9.3.1.5. Disabling or enabling the SR-IOV Operator admission controller webhook
9.3.1.6. Configuring a custom NodeSelector for the SR-IOV Network Config daemon
9.4. CONFIGURING AN SR-IOV NETWORK DEVICE
9.4.1. Automated discovery of SR-IOV network devices
9.4.2. Configuring SR-IOV network devices
9.4.3. Virtual function (VF) partitioning for SR-IOV devices
9.5. CONFIGURING A SR-IOV NETWORK ATTACHMENT
9.5.1. Configuring SR-IOV additional network
9.5.1.1. Configuration for ipam CNI plug-in
9.5.1.1.1. Static IP address assignment configuration example
9.5.1.1.2. Dynamic IP address assignment configuration example
9.5.1.2. Configuring static MAC and IP addresses on additional SR-IOV networks
9.6. ADDING A POD TO AN SR-IOV ADDITIONAL NETWORK
9.6.1. Adding a Pod to an additional network
9.6.2. Creating a non-uniform memory access (NUMA) aligned SR-IOV pod
9.7. USING HIGH PERFORMANCE MULTICAST
9.7.1. Configuring high performance multicast
9.7.2. Using an SR-IOV interface for multicast
9.8. USING VIRTUAL FUNCTIONS (VFS) WITH DPDK AND RDMA MODES
9.8.1. Examples of using virtual functions in DPDK and RDMA modes
9.8.1.1. Example use of virtual function (VF) in DPDK mode with Intel NICs
9.8.1.2. Example use of a virtual function in DPDK mode with Mellanox NICs
9.8.1.3. Example of a virtual function in RDMA mode with Mellanox NICs

CHAPTER 10. OPENSHIFT SDN DEFAULT CNI NETWORK PROVIDER

10.1. ABOUT THE OPENSHIFT SDN DEFAULT CNI NETWORK PROVIDER
10.2. CONFIGURING EGRESS IPS FOR A PROJECT
10.2.1. Egress IP address assignment for project egress traffic
10.2.1.1. Considerations when using automatically assigned egress IP addresses
10.2.1.2. Considerations when using manually assigned egress IP addresses
10.2.2. Configuring automatically assigned egress IP addresses for a namespace
10.2.3. Configuring manually assigned egress IP addresses for a namespace
10.3. CONFIGURING AN EGRESS FIREWALL TO CONTROL ACCESS TO EXTERNAL IP ADDRESSES
10.3.1. How an egress firewall works in a project
10.3.1.1. Limitations of an egress firewall
10.3.1.2. Matching order for egress network policy rules
10.3.1.3. How Domain Name Server (DNS) resolution works
10.3.2. EgressNetworkPolicy custom resource (CR) object
10.3.2.1. EgressNetworkPolicy rules
10.3.2.2. Example EgressNetworkPolicy CR object
10.3.3. Creating an egress firewall policy object
10.4. EDITING AN EGRESS FIREWALL FOR A PROJECT
10.4.1. Editing an EgressNetworkPolicy object
10.4.2. EgressNetworkPolicy custom resource (CR) object
10.4.2.1. EgressNetworkPolicy rules
10.4.2.2. Example EgressNetworkPolicy CR object
10.5. REMOVING AN EGRESS FIREWALL FROM A PROJECT
10.5.1. Removing an EgressNetworkPolicy object
10.6. ENABLING MULTICAST FOR A PROJECT
10.6.1. About multicast
10.6.2. Enabling multicast between Pods
10.7. DISABLING MULTICAST FOR A PROJECT
10.7.1. Disabling multicast between Pods
10.8. CONFIGURING NETWORK ISOLATION USING OPENSHIFT SDN
10.8.1. Joining projects
10.8.2. Isolating a project
10.8.3. Disabling network isolation for a project
10.9. CONFIGURING KUBE-PROXY
10.9.1. About iptables rules synchronization
10.9.2. Modifying the kube-proxy configuration
10.9.3. kube-proxy configuration parameters

CHAPTER 11. OVN-KUBERNETES DEFAULT CNI NETWORK PROVIDER
11.1. ABOUT THE OVN-KUBERNETES DEFAULT CONTAINER NETWORK INTERFACE (CNI) NETWORK PROVIDER
11.1.1. OVN-Kubernetes features
11.1.2. Supported default CNI network provider feature matrix
11.1.3. Exposed metrics for OVN-Kubernetes
11.2. ENABLING MULTICAST FOR A PROJECT
11.2.1. About multicast
11.2.2. Enabling multicast between Pods
11.3. DISABLING MULTICAST FOR A PROJECT
11.3.1. Disabling multicast between Pods

CHAPTER 12. CONFIGURING ROUTES
12.1. ROUTE CONFIGURATION
12.1.1. Configuring route timeouts
12.1.2. Enabling HTTP strict transport security
12.1.3. Troubleshooting throughput issues
12.1.4. Using cookies to keep route statefulness
12.1.4.1. Annotating a route with a cookie
12.1.5. Route–specific annotations
12.1.6. Configuring the route admission policy
12.2. SECURED ROUTES
12.2.1. Creating a re-encrypt route with a custom certificate
CHAPTER 1. UNDERSTANDING NETWORKING

Kubernetes ensures that Pods are able to network with each other, and allocates each Pod an IP address from an internal network. This ensures all containers within the Pod behave as if they were on the same host. 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.

1.1. OPENS SHIFT 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.
CHAPTER 2. ACCESSING HOSTS

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

2.1. 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. In order 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, for example, you can provide keys via Ignition, like the installer does.

4. Once 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 host name 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 master host. Ensure that you use the same SSH key you specified during the installation:

   ```
   $ ssh -i <ssh-key-path> core@<master-hostname>
   ```
CHAPTER 3. 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 default Container Network Interface (CNI) network provider plug-in selected for the cluster during installation.

3.1. CLUSTER NETWORK OPERATOR

The Cluster Network Operator implements the `network` API from the `operator.openshift.io` API group. The Operator deploys the OpenShift SDN default Container Network Interface (CNI) network provider plug-in, or the default network provider plug-in that you selected during cluster installation, by using a DaemonSet.

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
   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
   NAME   VERSION   AVAILABLE   PROGRESSING   DEGRADED   SINCE
   network 4.4.0     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.

3.2. 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
  Name:         cluster
  Namespace:    <none>
  Labels:       <none>
  Annotations:  <none>
  API Version:  config.openshift.io/v1
  Kind:         Network
  Metadata:
  ```
Self Link: /apis/config.openshift.io/v1/networks/cluster

Spec:
Cluster Network:
  Cidr: 10.128.0.0/14
  Host Prefix: 23
Network Type: OpenShiftSDN
Service Network: 172.30.0.0/16

Status:
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.

3.3. 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

3.4. 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
IMPORTANT

The Open Virtual Networking (OVN) Kubernetes network plug-in 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 the OVN Technology Preview, see https://access.redhat.com/articles/4380121.

3.5. 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 CR object that is named `cluster`. The CR specifies the parameters for the `Network` API in the `operator.openshift.io` API group.

You can specify the cluster network configuration for your OpenShift Container Platform cluster by setting the parameter values for the `defaultNetwork` parameter in the CNO CR. The following CR displays the default configuration for the CNO and explains both the parameters you can configure and the valid parameter values:

**Cluster Network Operator CR**

```yaml
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
  kubeProxyConfig: 4
    iptablesSyncPeriod: 30s 5
    proxyArguments:
      iptables-min-sync-period: 6
      - 30s
```

1. A list specifying the blocks of IP addresses from which Pod IPs are allocated and the subnet prefix length assigned to each individual node.

2. A block of IP addresses for services. The OpenShift SDN Container Network Interface (CNI) plug-in supports only a single IP address block for the service network.

3. Configures the default Container Network Interface (CNI) network provider for the cluster network.

4. The parameters for this object specify the Kubernetes network proxy (kube-proxy) configuration. If you are using the OVN-Kubernetes network provider, the kube-proxy configuration has no effect.
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.

The minimum duration before refreshing `iptables` rules. This parameter ensures that the refresh does not happen too frequently. Valid suffixes include s, m, and h and are described in the Go time package.

3.5.1. Configuration parameters for the OpenShift SDN default CNI network provider

The following YAML object describes the configuration parameters for the OpenShift SDN default Container Network Interface (CNI) network provider.

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

1. The default CNI network provider plug-in that is used.
2. OpenShift SDN specific configuration parameters.
3. The network isolation mode for OpenShift SDN.
4. The maximum transmission unit (MTU) for the VXLAN overlay network. This value is normally configured automatically.
5. The port to use for all VXLAN packets. The default value is 4789.

3.5.2. Configuration parameters for the OVN-Kubernetes default CNI network provider

The following YAML object describes the configuration parameters for the OVN-Kubernetes default CNI network provider.

```
defaultNetwork:
  type: OVNKubernetes
  ovnKubernetesConfig:
```

1. OpenShift Container Platform 4.4 Networking
mtu: 1450 3
genevePort: 6081 4

1. The default CNI network provider plug-in that is used.
2. OVN-Kubernetes specific configuration parameters.
3. The MTU for the Generic Network Virtualization Encapsulation (GENEVE) overlay network. This value is normally configured automatically.
4. The UDP port for the GENEVE overlay network.

3.5.3. Cluster Network Operator example configuration

A complete CR object for the CNO is displayed in the following example:

**Cluster Network Operator example CR**

```yaml
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  clusterNetwork:
    - cidr: 10.128.0.0/14 3
      hostPrefix: 23
  serviceNetwork:
    - 172.30.0.0/16
  defaultNetwork:
    type: OpenShiftSDN 4
    openshiftSDNConfig:
      mode: NetworkPolicy
      mtu: 1450
    vxlanPort: 4789
  kubeProxyConfig:
    iptablesSyncPeriod: 30s
    proxyArguments:
      - iptables-min-sync-period: 30s
```

CHAPTER 3. CLUSTER NETWORK OPERATOR IN OPENShift CONTAINER PLATFORM
CHAPTER 4. 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.

4.1. DNS OPERATOR

The DNS Operator implements the `dns` API from the `operator.openshift.io` API group. The operator deploys CoreDNS using a DaemonSet, creates a Service for the DaemonSet, and configures the kubelet to instruct pods to use the CoreDNS Service IP for name resolution.

Procedure

The DNS Operator is deployed during installation as a Kubernetes Deployment.

1. Use the `oc get` command to view the Deployment status:

   ```bash
   $ oc get -n openshift-dns-operator deployment/dns-operator
   NAME           READY     UP-TO-DATE   AVAILABLE   AGE
   dns-operator   1/1       1            1           23h
   ```

   ClusterOperator is the Custom Resource object which holds the current state of an operator. This object is used by operators to convey their state to the rest of the cluster.

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

   ```bash
   $ oc get clusteroperator/dns
   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 DaemonSet is reporting an Available status condition.

4.2. 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`:

   ```bash
   $ oc describe dns.operator/default
   Name:         default
   Namespace:    <none>
   Labels:       <none>
   Annotations:  <none>
   API Version:  operator.openshift.io/v1
   Kind:         DNS
   Status:
   ```
Cluster Domain: cluster.local
Cluster IP: 172.30.0.10

The Cluster Domain field is the base DNS domain used to construct fully qualified Pod and Service domain names.

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:

```bash
$ oc get networks.config/cluster -o jsonpath='{$.status.serviceNetwork}'
[172.30.0.0/16]
```

### 4.3. USING DNS FORWARDING

You can use DNS forwarding to override the forwarding configuration identified in `etc/resolv.conf` on a per-zone basis by specifying which name server should be used for a given zone.

**Procedure**

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

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

This allows the Operator to create and update the ConfigMap named **dns-default** with additional server configuration blocks based on Server. If none of the servers has a zone that matches the query, then name resolution falls back to the name servers that are specified in `/etc/resolv.conf`.

**Sample DNS**

```yaml
apiVersion: operator.openshift.io/v1
kind: DNS
metadata:
  name: default
spec:
servers:
- name: foo-server
  zones:
  - foo.com
forwardPlugin:
  upstreams:
  - 1.1.1.1
  - 2.2.2.2:5353
- name: bar-server
  zones:
  - bar.com
  - example.com
forwardPlugin:
```
name must comply with the rfc6335 service name syntax.

zones must conform to the definition of a subdomain in rfc1123. The cluster domain, cluster.local, is an invalid subdomain for zones.

A maximum of 15 upstreams is allowed per forwardPlugin.

NOTE

If servers is undefined or invalid, the ConfigMap only contains the default server.

2. View the ConfigMap:

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

Sample DNS ConfigMap based on previous sample DNS

```yaml
apiVersion: v1
data:
  Corefile: |
  foo.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
  }
  .: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 {
      policy sequential
    }
    cache 30
    reload
  }
kind: ConfigMap
metadata:
  labels:
    dns.operator.openshift.io/owning-dns: default
  name: dns-default
  namespace: openshift-dns
```

Changes to the forwardPlugin triggers a rolling update of the CoreDNS DaemonSet.
Additional resources

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

4.4. 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
```

4.5. 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 --namespace=openshift-dns-operator deployment/dns-operator
```
CHAPTER 5. INGRESS OPERATOR IN OPENSSHIFT CONTAINER PLATFORM

The Ingress Operator implements the ingresscontroller API and is the component responsible for enabling external access to OpenShift Container Platform cluster services. The Operator makes this possible 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.

5.1. 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

```yaml
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 as the domain used when generating a default host for a Route resource that does not specify an explicit host.

5.2. INGRESS CONTROLLER CONFIGURATION PARAMETERS

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>apiVersion</td>
<td>config.openshift.io/v1</td>
</tr>
<tr>
<td>kind</td>
<td>Ingress</td>
</tr>
<tr>
<td>metadata.name</td>
<td>cluster</td>
</tr>
<tr>
<td>spec.domain</td>
<td>apps.openshiftdemos.com</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>domain</td>
<td><strong>domain</strong> is a DNS name serviced by the Ingress controller and is used to configure multiple features:</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The <strong>domain</strong> value must be unique among all Ingress controllers and cannot be updated.</td>
</tr>
<tr>
<td></td>
<td>If empty, the default value is <strong>ingress.config.openshift.io/cluster.spec.domain</strong>.</td>
</tr>
<tr>
<td>replicas</td>
<td><strong>replicas</strong> is the desired number of Ingress controller replicas. If not set, the default value is 2.</td>
</tr>
<tr>
<td>endpointPublishingStrategy</td>
<td><strong>endpointPublishingStrategy</strong> is used to publish the Ingress controller endpoints to other networks, enable load balancer integrations, and provide access to other systems.</td>
</tr>
<tr>
<td></td>
<td>If not set, the default value is based on <strong>infrastructure.config.openshift.io/cluster.status.platform</strong>.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The <strong>endpointPublishingStrategy</strong> value cannot be updated.</td>
</tr>
<tr>
<td>defaultCertificate</td>
<td>The <strong>defaultCertificate</strong> 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, <strong>defaultCertificate</strong> is used.</td>
</tr>
<tr>
<td></td>
<td>The secret must contain the following keys and data:</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>If not set, a wildcard certificate is automatically generated and used. The certificate is valid for the Ingress controller <strong>domain</strong> and <strong>subdomains</strong>, and the generated certificate’s CA is automatically integrated with the cluster’s trust store.</td>
</tr>
<tr>
<td></td>
<td>The in-use certificate, whether generated or user-specified, is automatically integrated with OpenShift Container Platform built-in OAuth server.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>namespaceSelector</td>
<td>namespaceSelector is used to filter the set of namespaces serviced by the Ingress controller. This is useful for implementing shards.</td>
</tr>
<tr>
<td>routeSelector</td>
<td>routeSelector is used to filter the set of Routes serviced by the Ingress controller. This is useful for implementing shards.</td>
</tr>
<tr>
<td>nodePlacement</td>
<td>nodePlacement enables explicit control over the scheduling of the Ingress controller. If not set, the defaults values are used.</td>
</tr>
</tbody>
</table>

**NOTE**

The nodePlacement parameter includes two parts, nodeSelector and tolerations. For example:

```
nodePlacement:
  nodeSelector:
    matchLabels:
      beta.kubernetes.io/os: linux
  tolerations:
    - effect: NoSchedule
      operator: Exists
```
### Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tlsSecurityProfile</td>
<td>Specifies settings for TLS connections for Ingress controllers. If not set, the default value is based on the <code>apiservers.config.openshift.io/cluster</code> resource. When using the <code>Old</code>, <code>Intermediate</code>, and <code>Modern</code> profile types, the effective profile configuration is subject to change between releases. For example, given a specification to use the <code>Intermediate</code> 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 1.2. <strong>IMPORTANT</strong> The HAProxy Ingress controller image does not support TLS 1.3 and because the <code>Modern</code> profile requires TLS 1.3, it is not supported. The Ingress Operator converts the <code>Modern</code> profile to <code>Intermediate</code>. The Ingress Operator also converts the TLS 1.0 of an <code>Old</code> or <code>Custom</code> profile to 1.1, and TLS 1.3 of a <code>Custom</code> profile to 1.2. <strong>NOTE</strong> Ciphers and the minimum TLS version of the configured security profile are reflected in the <code>TLSProfile</code> status.</td>
</tr>
</tbody>
</table>
| routeAdmission  | Defines a policy for handling new route claims, such as allowing or denying claims across namespaces. `namespaceOwnership` describes how host name claims across namespaces should be handled. The default is `Strict`.  
- **Strict**: does not allow routes to claim the same host name across namespaces.  
- **InterNamespaceAllowed**: allows routes to claim different paths of the same host name across namespaces. **NOTE** All parameters are optional. |

### 5.2.1. Ingress controller TLS profiles

The `tlsSecurityProfile` parameter defines the schema for a TLS security profile. This object is used by operators to apply TLS security settings to operands.
There are four TLS security profile types:

- **Old**
- **Intermediate**
- **Modern**
- **Custom**

The **Old**, **Intermediate**, and **Modern** profiles are based on recommended configurations. The **Custom** profile provides the ability to specify individual TLS security profile parameters.

**Sample Old profile configuration**

```yaml
spec:
  tlsSecurityProfile:
    type: Old
```

**Sample Intermediate profile configuration**

```yaml
spec:
  tlsSecurityProfile:
    type: Intermediate
```

**Sample Modern profile configuration**

```yaml
spec:
  tlsSecurityProfile:
    type: Modern
```

The **Custom** profile is a user-defined TLS security profile.

**WARNING**

You must be careful using a **Custom** profile, because invalid configurations can cause problems.

**Sample Custom profile**

```yaml
spec:
  tlsSecurityProfile:
    type: Custom
    custom:
      ciphers:
        - ECDHE-ECDSA-AES128-GCM-SHA256
        - ECDHE-RSA-AES128-GCM-SHA256
      minTLSVersion: VersionTLS11
```
5.2.2. 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.

**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.

5.3. 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
  ```

5.4. 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

5.5. 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

5.6. VIEW INGRESS CONTROLLER STATUS

You 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>

5.7. 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 is valid for the Ingress domain.

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

$ oc --namespace openshift-ingress-operator get ingresscontrollers
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:
   
   ```
   $ oc get --namespace openshift-ingress-operator ingresscontrollers/default \  
   --output jsonpath='{.spec.defaultCertificate}'
   ```

   The output should look like:
   
   ```
   map[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.

### 5.8. 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`.

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

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

5.9. 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.

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: v1
items:
  - apiVersion: operator.openshift.io/v1
    kind: IngressController
    metadata:
      name: sharded
      namespace: openshift-ingress-operator
    spec:
      domain: <apps-sharded.basedomain.example.net>
      nodePlacement:
        nodeSelector:
          matchLabels:
            node-role.kubernetes.io/worker: ""
      routeSelector:
        matchLabels:
          type: sharded
    status: {}
  kind: List
  metadata:
    resourceVersion: ""
    selfLink: ""

2. Apply the Ingress Controller router-internal.yaml file:
The Ingress Controller selects routes in any namespace that have the label `type: sharded`.

**5.10. 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.

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:

   ```yaml
   # cat router-internal.yaml
   apiVersion: v1
   items:
   - apiVersion: operator.openshift.io/v1
     kind: IngressController
     metadata:
       name: sharded
       namespace: openshift-ingress-operator
     spec:
       domain: <apps-sharded.basedomain.example.net>
       nodePlacement:
         nodeSelector:
           matchLabels:
             node-role.kubernetes.io/worker: ""  
           namespaceSelector:
             matchLabels:
               type: sharded
       status: {}
   kind: List
   metadata:
   resourceVersion: ""
   selfLink: ""
   
   2. Apply the Ingress Controller `router-internal.yaml` file:

   ```bash
   # 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`.

**5.11. 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 object, you must delete and then recreate that IngressController object. You cannot change the .spec.endpointPublishingStrategy.loadBalancer.scope parameter after the Custom Resource (CR) is created.

See the Kubernetes Services documentation for implementation details.

**Prerequisites**

- Install the OpenShift command-line interface (CLI), commonly known as 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:

   ```yaml
   apiVersion: operator.openshift.io/v1
   kind: IngressController
   metadata:
     namespace: openshift-ingress-operator
     name: <name>  
   spec:
     domain: <domain>  
     endpointPublishingStrategy:
       type: LoadBalancerService
       loadBalancer:
         scope: Internal
   
   1. Replace `<name>` with a name for the IngressController object.
   2. Specify the domain for the application published by the controller.
   3. 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

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

### 5.12. 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** object, you must delete and then recreate that **IngressController** object. You cannot change the `.spec.endpointPublishingStrategy.loadBalancer.scope` parameter after the Custom Resource (CR) is created.

#### Prerequisites

- Install the OpenShift command-line interface (CLI), commonly known as `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
   ```

   ```
5.13. 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 host name.

**WARNING**

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

**Prerequisites**

- Cluster administrator privileges.

**Procedure**

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

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

**Sample Ingress Controller configuration**

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

5.14. ADDITIONAL RESOURCES

- Configuring a custom PKI
6.1. ABOUT NETWORK POLICY

In a cluster using a Kubernetes Container Network Interface (CNI) plug-in that supports Kubernetes network policy, network isolation is controlled entirely by NetworkPolicy Custom Resource (CR) objects. In OpenShift Container Platform 4.4, OpenShift SDN supports using NetworkPolicy in its default network isolation mode.

**NOTE**

The Kubernetes v1 NetworkPolicy features are available in OpenShift Container Platform except for egress policy types and IPBlock.

**WARNING**

Network policy does not apply to the host network namespace. Pods with host networking enabled are unaffected by NetworkPolicy object rules.

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.

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:

  ```yaml
  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:

  ```yaml
  apiVersion: networking.k8s.io/v1
  ```
If the Ingress Controller is configured with `endpointPublishingStrategy: HostNetwork`, then the Ingress Controller Pod runs on the host network. When running on the host network, the traffic from the Ingress Controller is assigned the `netid:0` Virtual Network ID (VNID). The `netid` for the namespace that is associated with the Ingress Operator is different, so the `matchLabel` in the `allow-from-openshift-ingress` network policy does not match traffic from the `default` Ingress Controller. Because the `default` namespace is assigned the `netid:0` VNID, you can allow traffic from the `default` Ingress Controller by labeling your `default` namespace with `network.openshift.io/policy-group: 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:

```yaml
kind: NetworkPolicy
metadata:
  name: allow-same-namespace
spec:
  ingress:
    - from:
      namespaceSelector:
        matchLabels:
          network.openshift.io/policy-group: ingress
      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:

```yaml
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:

```yaml
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.

### 6.2. EXAMPLE NETWORKPOLICY OBJECT

The following annotates an example NetworkPolicy object:

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

1. The `name` of the NetworkPolicy object.
2.
A selector describing the Pods the policy applies to. The policy object can only select Pods in the project that the NetworkPolicy object is defined.

A selector matching the Pods that the policy object allows ingress traffic from. The selector will match Pods in any project.

A list of one or more destination ports to accept traffic on.

### 6.3. CREATING A NETWORKPOLICY OBJECT

To define granular rules describing Ingress network traffic allowed for projects in your cluster, you can create NetworkPolicy objects.

**Prerequisites**

- A cluster using the OpenShift SDN network plug-in with *mode: NetworkPolicy* set. This mode is the default for OpenShift SDN.
- Install the OpenShift Command-line Interface (CLI), commonly known as *oc*.
- You must log in to the cluster.

**Procedure**

1. Create a policy rule:
   a. Create a `<policy-name>.yaml` file where `<policy-name>` describes the policy rule.
   b. In the file you just created define a policy object, such as in the following example:

   ```yaml
   kind: NetworkPolicy
   apiVersion: networking.k8s.io/v1
   metadata:
     name: <policy-name>  
   spec:
     podSelector:
     ingress: []
   ```

   1. Specify a name for the policy object.

2. Run the following command to create the policy object:

   ```sh
   $ oc create -f <policy-name>.yaml -n <project>
   ```

   In the following example, a new NetworkPolicy object is created in a project named `project1`:

   ```sh
   $ oc create -f default-deny.yaml -n project1
   networkpolicy "default-deny" created
   ```

### 6.4. DELETING A NETWORKPOLICY OBJECT

You can delete a NetworkPolicy object.
Prerequisites

- A cluster using the OpenShift SDN network plug-in with `mode: NetworkPolicy` set. This mode is the default for OpenShift SDN.
- Install the OpenShift Command-line Interface (CLI), commonly known as `oc`.
- You must log in to the cluster.

Procedure

- To delete a NetworkPolicy object, run the following command:

  ```bash
  $ oc delete networkpolicy -l name=<policy-name>  
  ```

  Specify the name of the NetworkPolicy object to delete.

6.5. VIEWING NETWORKPOLICY OBJECTS

You can list the NetworkPolicy objects in your cluster.

Prerequisites

- A cluster using the OpenShift SDN network plug-in with `mode: NetworkPolicy` set. This mode is the default for OpenShift SDN.
- Install the OpenShift Command-line Interface (CLI), commonly known as `oc`.
- You must log in to the cluster.

Procedure

- To view NetworkPolicy objects defined in your cluster, run the following command:

  ```bash
  $ oc get networkpolicy
  ```

6.6. CONFIGURING MULTITENANT ISOLATION USING NETWORKPOLICY

You can configure your project to isolate it from Pods and Services in other projects.

Prerequisites

- A cluster using the OpenShift SDN network plug-in with `mode: NetworkPolicy` set. This mode is the default for OpenShift SDN.
- Install the OpenShift Command-line Interface (CLI), commonly known as `oc`.
- You must log in to the cluster.

Procedure
1. Create the following files containing NetworkPolicy object definitions:

   a. A file named `allow-from-openshift-ingress.yaml` containing the following:

   ```yaml
   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
   ```

   b. A file named `allow-from-openshift-monitoring.yaml` containing the following:

   ```yaml
   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
   ```

2. For each policy file, run the following command to create the NetworkPolicy object:

   ```bash
   $ oc apply -f <policy-name>.yaml -n <project>
   ```

   Replace `<policy-name>` with the filename of the file containing the policy.

   Replace `<project>` with the name of the project to apply the NetworkPolicy object to.

3. If the `default` Ingress Controller configuration has the `spec.endpointPublishingStrategy: HostNetwork` value set, you must apply a label to the `default` OpenShift Container Platform namespace to allow network traffic between the Ingress Controller and the project:

   a. Determine if your `default` Ingress Controller uses the `HostNetwork` endpoint publishing strategy:

   ```bash
   $ oc get --namespace openshift-ingress-operator ingresscontrollers/default --output jsonpath=’.status.endpointPublishingStrategy.type’
   ```
b. If the previous command reports the endpoint publishing strategy as `HostNetwork`, set a label on the `default` namespace:

```bash
$ oc label namespace default 'network.openshift.io/policy-group=ingress'
```

4. Optional: Confirm that the NetworkPolicy object exists in your current project by running the following command:

```bash
$ oc get networkpolicy <policy-name> -o yaml
```

In the following example, the `allow-from-openshift-ingress` NetworkPolicy object is displayed:

```bash
$ oc get networkpolicy allow-from-openshift-ingress -o yaml
```

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-openshift-ingress
  namespace: project1
spec:
  ingress:
  - from:
    - namespaceSelector:
      matchLabels:
        network.openshift.io/policy-group: ingress
      podSelector: {}
  policyTypes:
  - Ingress
```

### 6.7. Creating Default Network Policies for a New Project

As a cluster administrator, you can modify the new project template to automatically include NetworkPolicy objects when you create a new project.

#### 6.7.1. 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:

   ```bash
   $ 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 Global 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.

6.7.2. Adding network policy objects to the new project template

As a cluster administrator, you can add network policy objects to the default template for new projects. OpenShift Container Platform will automatically create all the NetworkPolicy CRs specified in the template in the project.

**Prerequisites**

- A cluster using the OpenShift SDN network plug-in with mode: NetworkPolicy set. This mode is the default for OpenShift SDN.
- Install the OpenShift Command-line Interface (CLI), commonly known as 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:

   ```bash
   $ 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:

   ```yaml
   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
   ...
   ```

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:

   ```bash
   $ 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:

   ```bash
   $ oc get networkpolicy
   NAME                           POD-SELECTOR   AGE
   allow-from-openshift-ingress   <none>         7s
   allow-from-same-namespace      <none>         7s
   ```
CHAPTER 7. 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.

7.1. 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 must be defined with the type parameter set to either the ClusterIP or NodePort value.

7.1.1. 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 resource definition.

In the following example, a Pod is configured to use SCTP:

```yaml
apiVersion: v1
category: Pod
description: 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:

```yaml
apiVersion: v1
category: Service
description: 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:

```yaml
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-sctp-on-http
spec:
  podSelector:
    matchLabels:
      role: web
  ingress:
  - ports:
  - protocol: SCTP
    port: 80
```

## 7.2. 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 Command-line Interface (CLI), commonly known as 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:

```yaml
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
name: load-sctp-module
spec:
  config:
    ignition:
      version: 2.2.0
    storage:
      files:
      - contents:
        source: data:,
        verification: {}
        filesystem: root
        mode: 420
        path: /etc/modprobe.d/sctp-blacklist.conf
        - contents:
          source: data:text/plain;charset=utf-8,sctp
```
2. To create the MachineConfig object, enter the following command:

```bash
$ 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.

```bash
$ oc get nodes
```

### 7.3. 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 Command-line Interface (CLI), commonly known as `oc`.
- Access to the cluster as a user with the `cluster-admin` role.

**Procedure**

1. Create a Pod that runs 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: scctpserver
     labels:
       app: scctpserver
   spec:
     containers:
     - name: scctpserver
       image: registry.access.redhat.com/ubi8/ubi
       command: ["/bin/sh", ",-c"]
       args:
         ["dnf install -y nc && sleep inf"]
       ports:
       - containerPort: 30102
         name: scctpserver
         protocol: SCTP
   ```

   b. Create the Pod by entering the following command:

      ```bash
      $ oc create -f scctp-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:

   ```yaml
   apiVersion: v1
   kind: Service
   metadata:
     name: sctpservice
     labels:
       app: sctpserver
   spec:
     type: NodePort
     selector:
       app: sctpserver
     ports:
     - name: sctpservice
       protocol: SCTP
       port: 30102
       targetPort: 30102
   ```

   b. To create the Service, enter the following command:

   ```bash
   $ 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:

   ```yaml
   apiVersion: v1
   kind: Pod
   metadata:
     name: sctpclient
     labels:
       app: sctpclient
   spec:
     containers:
     - name: sctpclient
       image: registry.access.redhat.com/ubi8/ubi
       command: ["/bin/sh", ",-c"]
       args:
         ["dnf install -y nc & & sleep inf"]
   ```

   b. To create the Pod object, enter the following command:

   ```bash
   $ 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:

   ```bash
   $ 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 8. MULTIPLE NETWORKS

8.1. UNDERSTANDING MULTIPLE NETWORKS

In Kubernetes, container networking is delegated to networking plug-ins that implement the Container Network Interface (CNI).

OpenShift Container Platform uses the Multus CNI plug-in to allow chaining of CNI plug-ins. 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 plug-ins 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.

8.1.1. 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 in order 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 Custom Resource (CR) that has a `NetworkAttachmentDefinition` type. A CNI configuration inside each of these CRs defines how that interface is created.

8.1.2. Additional networks in OpenShift Container Platform

OpenShift Container Platform provides the following CNI plug-ins for creating additional networks in your cluster:

- **bridge**: Creating a bridge-based additional network allows Pods on the same host to communicate with each other and the host.

- **host-device**: Creating a host-device additional network allows Pods access to a physical Ethernet network device on the host system.

- **macvlan**: Creating a macvlan-based additional network allows 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.

- **ipvlan**: Creating an ipvlan-based additional network allows 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.

- **SR-IOV**: Creating an SR-IOV based additional network allows Pods to attach to a virtual function (VF) interface on SR-IOV capable hardware on the host system.

### 8.2. ATTACHING A POD TO AN ADDITIONAL NETWORK

As a cluster user you can attach a Pod to an additional network.

#### 8.2.1. 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.

**Prerequisites**

- The Pod must be in the same namespace as the additional network.
- Install the OpenShift Command-line Interface (CLI), commonly known as **oc**.
- You must log in to the cluster.

**Procedure**

To add a Pod with additional networks, complete the following steps:

1. Create the Pod resource definition and add the **k8s.v1.cni.cncf.io/networks** parameter to the Pod **metadata** mapping. The **k8s.v1.cni.cncf.io/networks** accepts a comma separated string of one or more NetworkAttachmentDefinition Custom Resource (CR) names:

```
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: <network>[,<network>,...]
```

Replace **<network>** with the name of the additional network to associate with the Pod. 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.

In the following example, two additional networks are attached to the Pod:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: example-pod
  annotations:
    k8s.v1.cni.cncf.io/networks: net1,net2
spec:
  containers:
  - name: example-pod
    command: ["/bin/bash", "-c", "sleep 2000000000000"]
  image: centos/tools
```
2. Create the Pod by running the following command:

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

3. Optional: Confirm that the annotation exists in the Pod CR by running the following command. Replace `<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/networks-status: |
      |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
        |
```
Prerequisites

- The Pod must be in the same namespace as the additional network.
- Install the OpenShift Command-line Interface (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, run the following command to edit its definition in the default editor. Replace `<name>` with the name of the Pod to edit.

   ```bash
   $ 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.

   ```yaml
   metadata:
     annotations:
       k8s.v1.cni.cncf.io/networks: '[<network>[,<network>,...]]'
   
   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.

   ```yaml
   apiVersion: v1
   kind: Pod
   metadata:
     name: example-pod
     annotations:
       k8s.v1.cni.cncf.io/networks: 
         
           "name": "net1"
   
           "name": "net2",
   
           "default-route": ["192.0.2.1"]
   
   spec:
     containers:
     - name: example-pod
       command: ["/bin/bash", "-c", "sleep 2000000000000"]
       image: centos/tools
   
   The `name` key is the name of the additional network to associate with the Pod.
   
   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.
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.

```bash
$ oc exec -it <pod_name> -- ip route
```

**NOTE**

You may also reference the Pod’s `k8s.v1.cni.cncf.io/networks-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:

```bash
$ oc edit networks.operator.openshift.io cluster
```

The following YAML describes the configuration parameters for the CNO:

**Cluster Network Operator YAML configuration**

```yaml
name: <name>  
namespace: <namespace>  
rawCNIConfig: '{  ...
   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 plug-in 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 plug-in:
macvlan CNI plug-in JSON configuration object using static IP and MAC address

```json
{
  "cniVersion": "0.3.1",
  "plugins": [{
    "type": "macvlan",
    "capabilities": { "ips": true },
    "master": "eth0",
    "mode": "bridge",
    "ipam": {
      "type": "static"
    }
  }],
  "capabilities": { "mac": true, "type": "tuning" }
}
```

1. The **plugins** field specifies a configuration list of CNI configurations.
2. The **capabilities** key denotes that a request is being made to enable the static IP functionality of a CNI plug-ins runtime configuration capabilities.
3. The **master** field is specific to the macvlan plug-in.
4. Here the **capabilities** key denotes that a request is made to enable the static MAC address functionality of a CNI plug-in.

The above network attachment may then 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 desired Pod with:

```
$ oc edit pod <name>
```

macvlan CNI plug-in JSON configuration object using static IP and MAC address

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: example-pod
annotations:
  k8s.v1.cni.cncf.io/networks: '[
    {
      "name": "<name>",
      "ips": [ "192.0.2.205/24" ],
      "mac": "CA:FE:C0:FF:EE:00"
    }
  ]
```

1. Use the `<name>` as provided when creating the `rawCNIConfig` above.
2. Provide the desired IP address.
3 Provide the desired 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

8.3. REMOVING A POD FROM AN ADDITIONAL NETWORK

As a cluster user you can remove a Pod from an additional network.

8.3.1. Removing a Pod from an additional network

You can remove a Pod from an additional network.

Prerequisites
- You have configured an additional network for your cluster.
- You have an additional network attached to the Pod.
- Install the OpenShift Command-line Interface (CLI), commonly known as oc.
- You must log in to the cluster.

Procedure
To remove a Pod from an additional network, complete the following steps:

1. Edit the Pod resource definition by running the following command. Replace <name> with the name of the Pod to edit.

   $ oc edit pod <name>

2. Update the annotations mapping to remove the additional network from the Pod by performing one of the following actions:

   - To remove all additional networks from a Pod, remove the k8s.v1.cni.cncf.io/networks parameter from the Pod resource definition as in the following example:

     ```
     apiVersion: v1
     kind: Pod
     metadata:
       name: example-pod
     annotations: {}
     spec:
       containers:
     ```
8.4. CONFIGURING A BRIDGE NETWORK

As a cluster administrator, you can configure an additional network for your cluster using the bridge Container Network Interface (CNI) plug-in. When configured, all Pods on a node are connected to a virtual switch. Each Pod is assigned an IP address on the additional network.

8.4.1. Creating an additional network attachment with the bridge CNI plug-in

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.

**IMPORTANT**

Do not edit the NetworkAttachmentDefinition CRs that the Cluster Network Operator manages. Doing so might disrupt network traffic on your additional network.

**Prerequisites**
• Install the OpenShift Command-line Interface (CLI), commonly known as **oc**.

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

**Procedure**

To create an additional network for your cluster, complete the following steps:

1. Edit the CNO CR by running the following command:

   ```sh
   $ oc edit networks.operator.openshift.io cluster
   ```

2. Modify the CR that you are creating by adding the configuration for the additional network you are creating, as in the following example CR.

   The following YAML configures the bridge CNI plug-in:

   ```yaml
   apiVersion: operator.openshift.io/v1
   kind: Network
   metadata:
     name: cluster
   spec:
     additionalNetworks:
     - name: test-network-1
       namespace: test-1
       type: Raw
       rawCNIConfig:
         "cniVersion": "0.3.1",
         "name": "test-network-1",
         "type": "bridge",
         "master": "eth1",
         "ipam": {
           "type": "static",
           "addresses": [
             {
               "address": "191.168.1.1/24"
             }
           ]
         }
   ```

   **1** Specify the configuration for the additional network attachment definition.

3. Save your changes and quit the text editor to commit your changes.

4. Optional: Confirm that the CNO created the NetworkAttachmentDefinition CR by running the following command. There might be a delay before the CNO creates the CR.

   ```sh
   $ oc get network-attachment-definitions -n <namespace>
   NAME        AGE
   test-network-1 14m
   ```

**8.4.1.1. Configuration for bridge**
The configuration for an additional network attachment that uses the bridge Container Network Interface (CNI) plug-in is provided in two parts:

- Cluster Network Operator (CNO) configuration
- CNI plug-in configuration

The CNO configuration specifies the name for the additional network attachment and the namespace to create the attachment in. The plug-in is configured by a JSON object specified by the `rawCNIConfig` parameter in the CNO configuration.

The following YAML describes the configuration parameters for the CNO:

**Cluster Network Operator YAML configuration**

```yaml
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 plug-in configuration in JSON format, which is based on the following template.

The following object describes the configuration parameters for the bridge CNI plug-in:

**bridge CNI plug-in JSON configuration object**

```json
{
  "cniVersion": "0.3.1",
  "name": "<name>",  # 1
  "type": "bridge",
  "bridge": "<bridge>",  # 2
  "ipam": {
    ...
  },
  "ipMasq": false,  # 4
  "isGateway": false,  # 5
  "isDefaultGateway": false,  # 6
  "forceAddress": false,  # 7
  "hairpinMode": false,  # 8
  "promiscMode": false,  # 9
  "vlan": <vlan>,  # 10
  "mtu": <mtu>  # 11
}
```

1. Specify the value for the `name` parameter you provided previously for the CNO configuration.
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`.

Specify a configuration object for the ipam CNI plug-in. The plug-in manages IP address assignment for the network attachment definition.

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`.

Set to `true` to assign an IP address to the bridge. The default value is `false`.

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.

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`.

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`.

Set to `true` to enable promiscuous mode on the bridge. The default value is `false`.

Specify a virtual LAN (VLAN) tag as an integer value. By default, no VLAN tag is assigned.

Set the maximum transmission unit (MTU) to the specified value. The default value is automatically set by the kernel.

8.4.1.1. bridge configuration example

The following example configures an additional network named `bridge-net`:

```yaml
name: bridge-net
namespace: work-network
type: Raw
rawCNIConfig: '{
    "cniVersion": "0.3.1",
    "name": "work-network",
    "type": "bridge",
    "master": "eth1",
    "isGateway": true,
    "vlan": 2,
    "ipam": {
        "type": "dhcp"
    }
}
```

The CNI configuration object is specified as a YAML string.

8.4.1.2. Configuration for ipam CNI plug-in

The IP address management (IPAM) Container Network Interface (CNI) plug-in manages IP address assignment for other CNI plug-ins. You can configure ipam for either static IP address assignment or
dynamic IP address assignment by using DHCP. The DHCP server you specify must be reachable from
the additional network.

The following JSON configuration object describes the parameters that you can set.

**IMPORTANT**

If you set the *type* parameter to the **DHCP** value, you cannot set any other parameters.

**ipam CNI plug-in JSON configuration object**

```
{
  "ipam": {
    "type": "<type>",
    "addresses": [
      {
        "address": "<address>",
        "gateway": "<gateway>",
      },
    ],
    "routes": [
      {
        "dst": "<dst>",
        "gw": "<gw>",
      },
    ],
    "dns": {
      "nameservers": ["<nameserver>"],
      "domain": "<domain>",
      "search": ["<search_domain>"]
    }
  }
}
```

1. Specify **static** to configure the plug-in to manage IP address assignment. Specify **DHCP** to allow a DHCP server to manage IP address assignment. You cannot specify any additional parameters if you specify a value of **DHCP**.

2. An array describing IP addresses to assign to the virtual interface. Both IPv4 and IPv6 IP addresses are supported.

3. A block of IP addresses that you specify in CIDR format to assign to Pods on a worker node, such as 10.1.1.0/24.

4. The default gateway to route egress network traffic to.

5. An array describing routes to configure inside the Pod.

6. The IP address range in CIDR format.

7. The gateway where network traffic is routed.

8. Optional: DNS configuration.
An array of one or more IP addresses for to send DNS queries to.

The default domain to append to a host name. For example, if the domain is set to `example.com`, a DNS lookup query for `example-host` is rewritten as `example-host.example.com`.

An array of domain names to append to an unqualified host name, such as `example-host`, during a DNS lookup query.

### 8.4.1.2.1. Static IP address assignment configuration example

You can configure ipam for static IP address assignment:

```json
{
   "ipam": {
      "type": "static",
      "addresses": [
         {
            "address": "191.168.1.1/24"
         }
      ]
   }
}
```

### 8.4.1.2.2. Dynamic IP address assignment configuration example

You can configure ipam for DHCP:

```json
{
   "ipam": {
      "type": "DHCP"
   }
}
```

**Next steps**

- Attach a Pod to an additional network.

### 8.5. Configuring a MACvlan network

As a cluster administrator, you can configure an additional network for your cluster using the macvlan CNI plug-in. When a Pod is attached to the network, the plug-in creates a sub-interface from the parent interface on the host. A unique hardware mac address is generated for each sub-device.

**IMPORTANT**

The unique MAC addresses this plug-in generates for sub-interfaces might not be compatible with the security polices of your cloud provider.

### 8.5.1. Creating an additional network attachment with the macvlan CNI plug-in
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.

**IMPORTANT**

Do not edit the NetworkAttachmentDefinition CRs that the Cluster Network Operator manages. Doing so might disrupt network traffic on your additional network.

**Prerequisites**

- Install the OpenShift Command-line Interface (CLI), commonly known as `oc`.
- Log in as a user with `cluster-admin` privileges.

**Procedure**

To create an additional network for your cluster, complete the following steps:

1. Edit the CNO CR by running the following command:

   ```bash
   $ oc edit networks.operator.openshift.io cluster
   ```

2. Modify the CR that you are creating by adding the configuration for the additional network you are creating, as in the following example CR. The following YAML configures the macvlan CNI plug-in:

   ```yaml
   apiVersion: operator.openshift.io/v1
   kind: Network
   metadata:
     name: cluster
   spec:
     additionalNetworks: 1
     - name: test-network-1
       namespace: test-1
       type: SimpleMacvlan
       simpleMacvlanConfig:
         ipamConfig:
           type: static
           staticIPAMConfig:
             addresses:
               - address: 10.1.1.0/24
   ```

   **Specify the configuration for the additional network attachment definition.**

3. Save your changes and quit the text editor to commit your changes.

4. Optional: Confirm that the CNO created the NetworkAttachmentDefinition CR by running the following command. There might be a delay before the CNO creates the CR.

   ```bash
   $ oc get network-attachment-definitions -n <namespace>
   NAME     AGE
   test-network-1 14m
   ```
8.5.1.1. Configuration for macvlan CNI plug-in

The following YAML describes the configuration parameters for the macvlan Container Network Interface (CNI) plug-in:

macvlan YAML configuration

```yaml
name: <name>  
namespace: <namespace>  
type: SimpleMacvlan  
simpleMacvlanConfig:  
  master: <master>  
  mode: <mode>  
  mtu: <mtu>  
ipamConfig:  
```

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 a value is not specified, the default namespace is used.

3. The ethernet interface to associate with the virtual interface. If a value for master is not specified, then the host system’s primary ethernet interface is used.

4. Configures traffic visibility on the virtual network. Must be either bridge, passthru, private, or vepa. If a value for mode is not provided, the default value is bridge.

5. Set the maximum transmission unit (MTU) to the specified value. The default value is automatically set by the kernel.

6. Specify a configuration object for the ipam CNI plug-in. The plug-in manages IP address assignment for the attachment definition.

8.5.1.1.1. macvlan configuration example

The following example configures an additional network named macvlan-net:

```yaml
name: macvlan-net  
namespace: work-network  
type: SimpleMacvlan  
simpleMacvlanConfig:  
ipamConfig:  
  type: DHCP  
```

8.5.1.2. Configuration for ipam CNI plug-in

The IP address management (IPAM) Container Network Interface (CNI) plug-in manages IP address assignment for other CNI plug-ins. You can configure ipam for either static IP address assignment or dynamic IP address assignment by using DHCP. The DHCP server you specify must be reachable from the additional network.
The following YAML configuration describes the parameters that you can set.

**IMPORTANT**

If you set the *type* parameter to the **DHCP** value, you cannot set any other parameters.

**ipam CNI plug-in YAML configuration object**

```yaml
ipamConfig:
  type: <type>
...```

1. Specify **static** to configure the plug-in to manage IP address assignment. Specify **DHCP** to allow a DHCP server to manage IP address assignment. You cannot specify any additional parameters if you specify a value of **DHCP**.

2. If you set the *type* parameter to **static**, then provide the **staticIPAMConfig** parameter.

**8.5.1.2.1. Static ipam configuration YAML**

The following YAML describes a configuration for static IP address assignment:

**Static ipam configuration YAML**

```yaml
ipamConfig:
  type: static
  staticIPAMConfig:
    addresses:
      - address: <address>
        gateway: <gateway>
    routes:
      - destination: <destination>
        gateway: <gateway>
    dns:
      nameservers:
        - <nameserver>
      domain: <domain>
      search:
        - <search_domain>
```

1. A collection of mappings that define IP addresses to assign to the virtual interface. Both IPv4 and IPv6 IP addresses are supported.

2. A block of IP addresses that you specify in CIDR format to assign to Pods on a worker node, such as 10.1.1.0/24.

3. The default gateway to route egress network traffic to.

4. A collection of mappings describing routes to configure inside the Pod.

5. The IP address range in CIDR format.
The gateway where network traffic is routed.

Optional: The DNS configuration.

A collection of one or more IP addresses for to send DNS queries to.

The default domain to append to a host name. For example, if the domain is set to example.com, a DNS lookup query for example-host is rewritten as example-host.example.com.

An array of domain names to append to an unqualified host name, such as example-host, during a DNS lookup query.

8.5.1.2.2. Dynamic ipam configuration YAML

The following YAML describes a configuration for static IP address assignment:

```
Dynamic ipam configuration YAML

ipamConfig:
  type: DHCP
```

8.5.1.2.3. Static IP address assignment configuration example

The following example shows an ipam configuration for static IP addresses:

```
ipamConfig:
  type: static
  staticIPAMConfig:
    addresses:
      - address: 198.51.100.11/24
        gateway: 198.51.100.10
    routes:
      - destination: 0.0.0.0/0
        gateway: 198.51.100.1
    dns:
      nameservers:
        - 198.51.100.1
        - 198.51.100.2
      domain: testDNS.example
      search:
        - testdomain1.example
        - testdomain2.example
```

8.5.1.2.4. Dynamic IP address assignment configuration example

The following example shows an ipam configuration for DHCP:

```
ipamConfig:
  type: DHCP
```

Next steps

- Attach a Pod to an additional network.
8.6. CONFIGURING AN IPVLAN NETWORK

As a cluster administrator, you can configure an additional network for your cluster by using the ipvlan Container Network Interface (CNI) plug-in. The virtual network created by this plug-in is associated with a physical interface that you specify.

8.6.1. Creating an additional network attachment with the ipvlan CNI plug-in

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.

IMPORTANT

Do not edit the NetworkAttachmentDefinition CRs that the Cluster Network Operator manages. Doing so might disrupt network traffic on your additional network.

Prerequisites

- Install the OpenShift Command-line Interface (CLI), commonly known as oc.
- Log in as a user with cluster-admin privileges.

Procedure

To create an additional network for your cluster, complete the following steps:

1. Edit the CNO CR by running the following command:

   $ oc edit networks.operator.openshift.io cluster

2. Modify the CR that you are creating by adding the configuration for the additional network you are creating, as in the following example CR.

   The following YAML configures the ipvlan CNI plug-in:

   ```yaml
   apiVersion: operator.openshift.io/v1
   kind: Network
   metadata:
     name: cluster
   spec:
     additionalNetworks: [1
       - name: test-network-1
         namespace: test-1
         type: Raw
         rawCNIConfig: {
           "cniVersion": "0.3.1",
           "name": "test-network-1",
           "type": "ipvlan",
           "master": "eth1",
           "mode": "l2",
           "ipam": {
             "type": "static",
             "addresses": [
               "191.168.1.1/24"
           ]
         }
       }
     ]
   ```
Specify the configuration for the additional network attachment definition.

3. Save your changes and quit the text editor to commit your changes.

4. Optional: Confirm that the CNO created the NetworkAttachmentDefinition CR by running the following command. There might be a delay before the CNO creates the CR.

```bash
$ oc get network-attachment-definitions -n <namespace>
NAME     AGE
test-network-1  14m
```

8.6.1.1. Configuration for ipvlan

The configuration for an additional network attachment that uses the ipvlan Container Network Interface (CNI) plug-in is provided in two parts:

- Cluster Network Operator (CNO) configuration
- CNI plug-in configuration

The CNO configuration specifies the name for the additional network attachment and the namespace to create the attachment in. The plug-in is configured by a JSON object specified by the `rawCNIConfig` parameter in the CNO configuration.

The following YAML describes the configuration parameters for the CNO:

**Cluster Network Operator YAML configuration**

```yaml
name: <name>  
namespace: <namespace>  
rawCNIConfig: '{
...
}'
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 plug-in configuration in JSON format, which is based on the following template.

The following object describes the configuration parameters for the ipvlan CNI plug-in:

**ipvlan CNI plug-in JSON configuration object**

```json
{
```
Specify the value for the name parameter you provided previously for the CNO configuration.

Specify the operating mode for the virtual network. The value must be \( l2, l3, \) or \( l3s \). The default value is \( l2 \).

Specify the ethernet interface to associate with the network attachment. If a master is not specified, the interface for the default network route is used.

Set the maximum transmission unit (MTU) to the specified value. The default value is automatically set by the kernel.

Specify a configuration object for the ipam CNI plug-in. The plug-in manages IP address assignment for the attachment definition.

### 8.6.1.1. ipvlan configuration example

The following example configures an additional network named ipvlan-net:

```yaml
name: ipvlan-net
namespace: work-network
type: Raw
rawCNIConfig: '{
  "cniVersion": "0.3.1",
  "name": "work-network",
  "type": "ipvlan",
  "master": "eth1",
  "mode": "l3",
  "ipam": {
    "type": "dhcp"
  }
}'
```

The CNI configuration object is specified as a YAML string.

### 8.6.1.2. Configuration for ipam CNI plug-in

The IP address management (IPAM) Container Network Interface (CNI) plug-in manages IP address assignment for other CNI plug-ins. You can configure ipam for either static IP address assignment or dynamic IP address assignment by using DHCP. The DHCP server you specify must be reachable from the additional network.

The following JSON configuration object describes the parameters that you can set.
IMPORTANT
If you set the type parameter to the DHCP value, you cannot set any other parameters.

ipam CNI plug-in JSON configuration object

```json
{
    "ipam": {
        "type": "<type>", 1
        "addresses": [
            {
                "address": "<address>", 3
                "gateway": "<gateway>" 4
            }
        ],
        "routes": [
            {
                "dst": "<dst>" 6
                "gw": "<gw>" 7
            }
        ],
        "dns": {
            "nameservers": [<nameserver>], 9
            "domain": "<domain>", 10
            "search": [<search_domain>] 11
        }
    }
}
```

1. Specify static to configure the plug-in to manage IP address assignment. Specify DHCP to allow a DHCP server to manage IP address assignment. You cannot specify any additional parameters if you specify a value of DHCP.

2. An array describing IP addresses to assign to the virtual interface. Both IPv4 and IPv6 IP addresses are supported.

3. A block of IP addresses that you specify in CIDR format to assign to Pods on a worker node, such as 10.1.1.0/24.

4. The default gateway to route egress network traffic to.

5. An array describing routes to configure inside the Pod.

6. The IP address range in CIDR format.

7. The gateway where network traffic is routed.

8. Optional: DNS configuration.

9. An array of one or more IP addresses for to send DNS queries to.

10. The default domain to append to a host name. For example, if the domain is set to example.com, a DNS lookup query for example-host is rewritten as example-host.example.com.
An array of domain names to append to an unqualified host name, such as `example-host`, during a DNS lookup query.

### 8.6.1.2.1. Static IP address assignment configuration example

You can configure ipam for static IP address assignment:

```json
{
  "ipam": {
    "type": "static",
    "addresses": [
      {"address": "191.168.1.1/24"}
    ]
  }
}
```

### 8.6.1.2.2. Dynamic IP address assignment configuration example

You can configure ipam for DHCP:

```json
{
  "ipam": {
    "type": "DHCP"
  }
}
```

#### Next steps
- Attach a Pod to an additional network.

### 8.7. CONFIGURING A HOST-DEVICE NETWORK

As a cluster administrator, you can configure an additional network for your cluster by using the host-device Container Network Interface (CNI) plug-in. The plug-in allows you to move the specified network device from the host's network namespace into the Pod's network namespace.

#### 8.7.1. Creating an additional network attachment with the host-device CNI plug-in

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.

**IMPORTANT**

Do not edit the NetworkAttachmentDefinition CRs that the Cluster Network Operator manages. Doing so might disrupt network traffic on your additional network.

**Prerequisites**

- Install the OpenShift Command-line Interface (CLI), commonly known as `oc`.
Log in as a user with `cluster-admin` privileges.

**Procedure**

To create an additional network for your cluster, complete the following steps:

1. Edit the CNO CR by running the following command:

   ```
   $ oc edit networks.operator.openshift.io cluster
   ```

2. Modify the CR that you are creating by adding the configuration for the additional network you are creating, as in the following example CR.

   The following YAML configures the host-device CNI plug-in:

   ```yaml
   apiVersion: operator.openshift.io/v1
   kind: Network
   metadata:
     name: cluster
   spec:
     additionalNetworks: 1
     - name: test-network-1
       namespace: test-1
       type: Raw
       rawCNIConfig: '{
         "cniVersion": "0.3.1",
         "name": "test-network-1",
         "type": "host-device",
         "device": "eth1"
       }'
   ```

   Specify the configuration for the additional network attachment definition.

3. Save your changes and quit the text editor to commit your changes.

4. Optional: Confirm that the CNO created the NetworkAttachmentDefinition CR by running the following command. There might be a delay before the CNO creates the CR.

   ```
   $ oc get network-attachment-definitions -n <namespace>
   NAME               AGE
   test-network-1     14m
   ```

**8.7.1.1. Configuration for host-device**

The configuration for an additional network attachment that uses the host-device Container Network Interface (CNI) plug-in is provided in two parts:

- Cluster Network Operator (CNO) configuration
- CNI plug-in configuration

The CNO configuration specifies the name for the additional network attachment and the namespace to create the attachment in. The plug-in is configured by a JSON object specified by the `rawCNIConfig` parameter in the CNO configuration.
The following YAML describes the configuration parameters for the CNO:

**Cluster Network Operator YAML configuration**

```yaml
name: <name>  
namespace: <namespace>  
rawCNIConfig: '{  
    ...  
}  
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 plug-in configuration in JSON format, which is based on the following template.

   **IMPORTANT**

   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 plug-in:

**host-device CNI plug-in JSON configuration object**

```json
{
    "cniVersion": "0.3.1",
    "name": "<name>",  
    "type": "host-device",
    "device": "<device>",
    "hwaddr": "<hwaddr>",
    "kernelpath": "<kernelpath>",
    "pciBusID": "<pciBusID>",
    "ipam": {  
        ...  
    }
}
```

1. Specify the value for the `name` parameter you provided previously for the CNO configuration.

2. Specify the name of the device, such as `eth0`.

3. Specify the device hardware MAC address.

4. Specify the Linux kernel device path, such as `/sys/devices/pci0000:00/0000:00:1f.6`.

5. Specify the PCI address of the network device, such as `0000:00:1f.6`.

6. Specify a configuration object for the `ipam` CNI plug-in. The plug-in manages IP address assignment for the attachment definition.
8.7.1.1.1. host-device configuration example

The following example configures an additional network named `hostdev-net`:

```markdown
name: hostdev-net
namespace: work-network
type: Raw
rawCNIConfig: '{
  "cniVersion": "0.3.1",
  "name": "work-network",
  "type": "host-device",
  "device": "eth1"
}'
```

1. The CNI configuration object is specified as a YAML string.

8.7.1.2. Configuration for ipam CNI plug-in

The IP address management (IPAM) Container Network Interface (CNI) plug-in manages IP address assignment for other CNI plug-ins. You can configure ipam for either static IP address assignment or dynamic IP address assignment by using DHCP. The DHCP server you specify must be reachable from the additional network.

The following JSON configuration object describes the parameters that you can set.

**IMPORTANT**

If you set the `type` parameter to the **DHCP** value, you cannot set any other parameters.

**ipam CNI plug-in JSON configuration object**

```json
{
  "ipam": {
    "type": "<type>",
    "addresses": [
      {
        "address": "<address>",
        "gateway": "<gateway>"
      }
    ],
    "routes": [
      {
        "dst": "<dst>",
        "gw": "<gw>"
      }
    ],
    "dns": {
      "nameservers": [<nameserver>],
      "domain": "<domain>",
      "search": [<search_domain>]"```
Specify **static** to configure the plug-in to manage IP address assignment. Specify **DHCP** to allow a DHCP server to manage IP address assignment. You cannot specify any additional parameters if you specify a value of **DHCP**.

An array describing IP addresses to assign to the virtual interface. Both IPv4 and IPv6 IP addresses are supported.

A block of IP addresses that you specify in CIDR format to assign to Pods on a worker node, such as **10.1.1.0/24**.

The default gateway to route egress network traffic to.

An array describing routes to configure inside the Pod.

The IP address range in CIDR format.

The gateway where network traffic is routed.

Optional: DNS configuration.

An array of one or more IP addresses for to send DNS queries to.

The default domain to append to a host name. For example, if the domain is set to **example.com**, a DNS lookup query for **example-host** is rewritten as **example-host.example.com**.

An array of domain names to append to an unqualified host name, such as **example-host**, during a DNS lookup query.

### 8.7.1.2.1. Static IP address assignment configuration example

You can configure ipam for static IP address assignment:

```json
{
  "ipam": {
    "type": "static",
    "addresses": [
      {
        "address": "191.168.1.1/24"
      }
    ]
  }
}
```

### 8.7.1.2.2. Dynamic IP address assignment configuration example

You can configure ipam for DHCP:

```json
{
  "ipam": {
    "type": "DHCP"
  }
}
```
8.8. EDITING AN ADDITIONAL NETWORK

As a cluster administrator you can modify the configuration for an existing additional network.

8.8.1. 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 Command-line Interface (CLI), commonly known as 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 CR 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 CR to reflect your changes.

   ```
   $ oc get network-attachment-definitions <network-name> -o yaml
   ```

For example, the following console output displays a NetworkAttachmentDefinition that is named net1:

   ```
   $ oc get network-attachment-definitions net1 -o go-template="{{printf "%s
" .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":
   }
   ```
8.9. REMOVING AN ADDITIONAL NETWORK

As a cluster administrator you can remove an additional network attachment.

8.9.1. 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 Command-line Interface (CLI), commonly known as `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: []
   ```

   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
   ```

8.10. CONFIGURING PTP
IMPORTANT

Precision Time Protocol (PTP) hardware 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 https://access.redhat.com/support/offerings/techpreview/.

8.10.1. About PTP hardware on OpenShift Container Platform

OpenShift Container Platform includes the capability to use PTP hardware on your nodes. You can configure linuxptp services on nodes with PTP capable hardware.

You can use the OpenShift Container Platform console to install PTP by deploying the PTP Operator. The PTP Operator creates and manages the linuxptp services. The Operator provides following features:

- Discover the PTP capable device in cluster.
- Manage configuration of linuxptp services.

8.10.2. Installing the PTP Operator

As a cluster administrator, you can install the PTP Operator using the OpenShift Container Platform CLI or the web console.

8.10.2.1. Installing the Operator using the CLI

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 PTP.
- The OpenShift Container Platform Command-line Interface (CLI), commonly known as oc.
- Log in as a user with cluster-admin privileges.

**Procedure**

1. Create a namespace for the PTP Operator by completing the following actions:
   a. Create the following Namespace Custom Resource (CR) that defines the openshift-tp namespace, and then save the YAML in the ptp-namespace.yaml file:

```
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-tp
labels:
  openshift.io/run-level: "1"
```
b. Create the namespace by running the following command:

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

2. Install the PTP Operator in the namespace you created in the previous step by creating the following objects:

a. Create the following OperatorGroup CR and save the YAML in the `ptp-operatorgroup.yaml` file:

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

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

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

c. Run the following command to get the `channel` value required for the next step.

```
$ oc get packagemanifest ptp-operator -n openshift-marketplace -o jsonpath='{.status.defaultChannel}'
```

4.4

d. Create the following Subscription CR and save the YAML in the `ptp-sub.yaml` file:

**Example Subscription**

```yaml
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: ptp-operator-subscription
  namespace: openshift-ptp
spec:
  channel: <channel> 1
  name: ptp-operator
  source: redhat-operators 2
  sourceNamespace: openshift-marketplace
```

1 Specify the value from you obtained in the previous step for the `.status.defaultChannel` parameter.

2 You must specify the `redhat-operators` value.

e. Create the Subscription object by running the following command:

```
$ oc create -f ptp-sub.yaml
```
f. Change to the `openshift-tp` project:

```
$ oc project openshift-tp
Now using project "openshift-tp"
```

### 8.10.2.2. Installing the Operator using the web console

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

**NOTE**

You have to create the Namespace CR and OperatorGroup CR 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 **Create Operator Subscription** page, under **A specific namespace on the cluster** select **openshift-tp**. Then, click **Subscribe**.

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-tp** 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-tp** project.

### 8.10.3. Automated discovery of PTP network devices

The PTP Operator adds the **NodePtpDevice.ptp.openshift.io** Custom Resource Definition (CRD) to OpenShift Container Platform. The PTP Operator will search your cluster for PTP capable network devices on each node. The Operator creates and updates a **NodePtpDevice** Custom Resource (CR) for each node that provides a compatible PTP device.
One CR is created for each node, and shares the same name as the node. The `status.devices` list provides information about the PTP devices on a node.

The following is an example of a NodePtpDevice CR created by the PTP Operator:

```yaml
apiVersion: ptp.openshift.io/v1
class: NodePtpDevice
metadata:
  creationTimestamp: "2019-11-15T08:57:11Z"
generation: 1
  name: dev-worker-0  
  namespace: openshift-tp
resourceVersion: "487462"
selfLink: /apis/ptp.openshift.io/v1/namespaces/openshift-tp/nodeptpdevices/dev-worker-0
uid: 08d133f7-aae2-403f-84ad-1fe624e5ab3f
spec: {}
status:
  devices:
  - name: eno1
  - name: eno2
  - name: ens787f0
  - name: ens787f1
  - name: ens801f0
  - name: ens801f1
  - name: ens802f0
  - name: ens802f1
  - name: ens803
```

1. The value for the `name` parameter is the same as the name of the node.
2. The CR is created in `openshift-tp` namespace by PTP Operator.
3. The `devices` collection includes a list of all of the PTP capable devices discovered by the Operator on the node.

### 8.10.4. Configuring Linuxptp services

The PTP Operator adds the `PtpConfig.ptp.openshift.io` Custom Resource Definition (CRD) to OpenShift Container Platform. You can configure the Linuxptp services (ptp4l, phc2sys) by creating a `PtpConfig` Custom Resource (CR).

#### Prerequisites

- Install the OpenShift Command-line Interface (CLI), commonly known as `oc`.
- Log in as a user with `cluster-admin` privileges.
- You must have installed the PTP Operator.

#### Procedure

1. Create the following `PtpConfig` CR, and then save the YAML in the `<name>-ptp-config.yaml` file. Replace `<name>` with the name for this configuration.
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: <name>  
  namespace: openshift-tp
spec:
  profile:
  - name: "profile1"
    interface: "ens787f1"
    ptp4lOpts: "-s -2"
    phc2sysOpts: "-a -r"
  recommend:
  - profile: "profile1"
    priority: 10
    match:
      - nodeLabel: "node-role.kubernetes.io/worker"
      - nodeName: "dev-worker-0"

1. Specify a name for the **PtpConfig** CR.
2. Specify the namespace where the PTP Operator is installed.
3. Specify an array of one or more **profile** objects.
4. Specify the name of a profile object which is used to uniquely identify a profile object.
5. Specify the network interface name to use by the **ptp4l** service, for example *ens787f1*.
6. Specify system config options for the **ptp4l** service, for example *-s -2*. This should not include the interface name *-i <interface>* and service config file *-f /etc/ptp4l.conf* because these will be automatically appended.
7. Specify system config options for the **phc2sys** service, for example *-a -r*.
8. Specify an array of one or more **recommend** objects which define rules on how the **profile** should be applied to nodes.
9. Specify the **profile** object name defined in the **profile** section.
10. 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 will be applied to that node.
11. Specify **match** rules with **nodeLabel** or **nodeName**.
12. Specify **nodeLabel** with the key of **node.Labels** from the node object.
13. Specify **nodeName** with **node.Name** from the node object.

2. Create the CR by running the following command:

    $ oc create -f <filename>  

### CHAPTER 8. MULTIPLE NETWORKS
Replace `<filename>` with the name of the file you created in the previous step.

3. Optional: Check that the **PtpConfig** profile is applied to nodes that match with **nodeLabel** or **nodeName**.

```
$ oc get pods -n openshift-ptp -o wide
NAME                      READY STATUS    RESTARTS AGE   IP               NODE
NOMINATED NODE           READY STATUS    RESTARTS AGE   IP               NODE
linuxptp-daemon-4xkbb    1/1   Running   0 43m   192.168.111.15 dev-worker-0
<none>                   <none>
linuxptp-daemon-tdspf    1/1   Running   0 43m   192.168.111.11 dev-master-0
<none>                   <none>
ptp-operator-657bb64c8-2f8sj  1/1   Running   0 43m   10.128.0.116 dev-master-0
<none>                   <none>
```

```
$ oc logs linuxptp-daemon-4xkbb -n openshift-ptp
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 09:41:17.117623 4143292 daemon.go:102 Phc2sysOpts: -a -r
I1115 09:41:17.117626 4143292 daemon.go:116 ------------------------------------
```

```
Profile Name is the name that is applied to node **dev-worker-0**.

**Interface** is the PTP device specified in the **profile1** interface field. The **ptp4l** service runs on this interface.

**Ptp4lOpts** are the ptp4l sysconfig options specified in **profile1** Ptp4lOpts field.

**Phc2sysOpts** are the phc2sys sysconfig options specified in **profile1** Phc2sysOpts field.

OpenShift Container Platform 4.4 Networking
CHAPTER 9. HARDWARE NETWORKS

9.1. ABOUT SINGLE ROOT I/O VIRTUALIZATION (SR-IOV) HARDWARE NETWORKS

You can use Single Root I/O Virtualization (SR-IOV) network devices with additional networks on your OpenShift Container Platform cluster for high performance applications.

9.1.1. About SR-IOV hardware on OpenShift Container Platform

OpenShift Container Platform includes the capability to use SR-IOV hardware on your nodes. You can attach SR-IOV virtual function (VF) interfaces to Pods on nodes with SR-IOV hardware.

You can use the OpenShift Container Platform console to install SR-IOV by deploying the SR-IOV Network Operator. The SR-IOV Network Operator creates and manages the components of the SR-IOV stack. The Operator provisions the following components:

- Provision the SR-IOV network operator deployment on master nodes.
- Provision the SR-IOV network config daemon on worker nodes.
- Provision the Operator webhook on master nodes.
- Provision the Network resources injector on master nodes.
- Provision the SR-IOV network device plug-in on worker nodes.
- Provision the SR-IOV CNI plug-in executable on worker nodes.

Here’s the function of each above mentioned SR-IOV components.

- The SR-IOV Operator is a Kubernetes Deployment that manages all SR-IOV components in a cluster. It watches creation, update and deletion of Operator Custom Resources and takes corresponding actions such as generating NetworkAttachmentDefinition Custom Resources for SR-IOV CNI, creating and updating configuration of SR-IOV network device plug-in, creating node specific SriovNetworkNodeState Custom Resources and updating Spec.Interfaces field in each SriovNetworkNodeState Custom Resource, etc.

- The SR-IOV network config daemon is a Kubernetes DaemonSet deployed on worker nodes when SR-IOV Operator is launched. It is responsible for discovering and initializing SR-IOV network devices in cluster.

- The Operator webhook is a Kubernetes Dynamic Admission Controller Webhook that validates correctness of Operator Custom Resource and sets default values for fields in Operator Custom Resource that are not configured by user.

- The Network resources injector is a Kubernetes 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 device plug-in is a Kubernetes device plug-in for discovering, advertising, and allocating SR-IOV network virtual function (VF) resources. Device plug-ins are used in Kubernetes to enable the use of limited resources, typically in physical devices. Device plug-ins give the Kubernetes scheduler awareness of resource availability, so the scheduler can schedule Pods on nodes with sufficient resources.
The SR-IOV CNI plug-in plumbs VF interfaces allocated from the SR-IOV device plug-in directly into a Pod.

NOTE

The Network resources injector and Operator webhook are enabled by default and can be disabled by editing the default SriovOperatorConfig CR.

9.1.1.1. Supported devices

The following Network Interface Card (NIC) models are supported in OpenShift Container Platform:

- Intel XXV710 25GbE SFP28 with vendor ID 0x8086 and device ID 0x158b
- Mellanox MT27710 Family [ConnectX-4 Lx] 25GbE dual-port SFP28 with vendor ID 0x15b3 and device ID 0x1015
- Mellanox MT27800 Family [ConnectX-5] 25GbE dual-port SFP28 with vendor ID 0x15b3 and device ID 0x1017
- Mellanox MT27800 Family [ConnectX-5] 100GbE with vendor ID 0x15b3 and device ID 0x1017

9.1.1.2. Example use of a virtual function (VF) 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 VF in RDMA mode:

```yaml
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:
        capabilities:
          add: ["IPC_LOCK"]
      command: ["sleep", "infinity"]
```

The following example shows a Pod with a VF in DPDK mode:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: dpdk-app
  annotations:
    k8s.v1.cni.cncf.io/networks: sriov-dpdk-net
spec:
  containers:
```

80
An optional library is available to aid the application running in a container in gathering network information associated with a pod. This library is called 'app-netutil'. See the library's source code in the app-netutil GitHub repo.

This library is intended to ease the integration of the SR-IOV VFs in DPDK mode into the container. The library provides both a GO API and a C API, as well as examples of using both languages.

There is also a sample Docker image, 'dpdk-app-centos', which can run one of the following DPDK sample applications based on an environmental variable in the pod-spec: l2fwd, l3wd or testpmd. This Docker image provides an example of integrating the 'app-netutil' into the container image itself. The library can also integrate into an init-container which collects the desired data and passes the data to an existing DPDK workload.

Next steps

- Installing the SR-IOV Network Operator
- Optional: Configuring the SR-IOV Network Operator
- Configuring an SR-IOV network device
- Configuring an SR-IOV network attachment
- Adding a Pod to an SR-IOV additional network

9.2. 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.

9.2.1. Installing SR-IOV Network Operator
As a cluster administrator, you can install the SR-IOV Network Operator using the OpenShift Container Platform CLI or the web console.

### 9.2.1.1. Installing the Operator using the CLI

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.
- The OpenShift Container Platform Command-line Interface (CLI), commonly known as `oc`.
- Log in as a user with `cluster-admin` privileges.

**Procedure**

1. Create a namespace for the SR-IOV Network Operator by completing the following actions:
   
   a. Create the following Namespace Custom Resource (CR) that defines the `openshift-sriov-network-operator` namespace, and then save the YAML in the `sriov-namespace.yaml` file:

   ```yaml
   apiVersion: v1
   kind: Namespace
   metadata:
     name: openshift-sriov-network-operator
     labels:
       openshift.io/run-level: "1"
   ```

   b. Create the namespace by running the following command:

   ```bash
   $ oc create -f sriov-namespace.yaml
   ```

2. Install the SR-IOV Network Operator in the namespace you created in the previous step by creating the following objects:

   a. Create the following OperatorGroup CR and save the YAML in the `sriov-operatorgroup.yaml` file:

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

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

   ```bash
   $ oc create -f sriov-operatorgroup.yaml
   ```

   c. Run the following command to get the `channel` value required for the next step.

   ```bash
   ```
$ oc get packagemanifest sriov-network-operator -n openshift-marketplace -o jsonpath='{.status.channels[].name}'

alpha

d. Create the following Subscription CR and save the YAML in the `sriov-sub.yaml` file:

Example Subscription

```yaml
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: sriov-network-operator-subscription
  namespace: openshift-sriov-network-operator
spec:
  channel: <channel> ¹
  name: sriov-network-operator
  source: redhat-operators ²
  sourceNamespace: openshift-marketplace
```

¹ Specify the value from you obtained in the previous step for the `.status.channels[].name` parameter.

² You must specify the `redhat-operators` value.

e. Create the Subscription object by running the following command:

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

f. Change to the `openshift-sriov-network-operator` project:

```
$ oc project openshift-sriov-network-operator

Now using project "openshift-sriov-network-operator"
```

9.2.1.2. Installing the Operator using the web console

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

**NOTE**

You have to create the Namespace CR and OperatorGroup CR as mentioned in the previous section.

**Procedure**

1. Install the SR-IOV Network Operator using the OpenShift Container Platform web console:
   a. In the OpenShift Container Platform web console, click **Operators → OperatorHub**.
   b. Choose **SR-IOV Network Operator** from the list of available Operators, and then click **Install**.

2. Optional: Verify that the SR-IOV Network Operator installed successfully:
   a. Switch 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:

- 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-sriov-network-operator project.

**Next steps**
- Optional: Configuring the SR-IOV Network Operator

### 9.3. CONFIGURING THE SR-IOV NETWORK OPERATOR


#### 9.3.1. 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.


**NOTE**

The default CR contains the SR-IOV Network Operator configuration for your cluster. To change the operator configuration, you must modify this CR.

The SriovOperatorConfig CR provides several fields for configuring the operator:
- **enableInjector** allows project administrators to enable or disable the Network Resources Injector DaemonSet.

- **enableOperatorWebhook** allows project administrators to enable or disable the Operator Admission Controller webhook DaemonSet.

- **configDaemonNodeSelector** allows project administrators to schedule the SR-IOV Network Config Daemon on selected nodes.

### 9.3.1.1. 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 Pod specification to add an SR-IOV resource name according to an SR-IOV network attachment definition annotation.

- Mutation of Pod specifications with downward API volume to expose pod annotations and labels to the running container as files under the `/etc/podnetinfo` path.

By default the Network Resources Injector is enabled by the SR-IOV operator and runs as a DaemonSet on all master nodes. The following is an example of Network Resources Injector Pods running in a cluster with three master nodes:

```
$ oc get pods -n openshift-sriov-network-operator
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
```

### 9.3.1.2. About the SR-IOV Operator admission controller webhook

The SR-IOV 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 Operator Admission Controller webhook is enabled by the operator and runs as a DaemonSet on all master nodes. The following is an example of the Operator Admission Controller webhook Pods running in a cluster with three master nodes:

```
$ oc get pods -n openshift-sriov-network-operator
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
```

### 9.3.1.3. 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.
9.3.1.4. 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 Command-line Interface (CLI), commonly known as **oc**.
- Log in as a user with **cluster-admin** privileges.
- You must have installed the SR-IOV Operator.

**Procedure**

- Set the **enableInjector** field. Replace `<value>` with **false** to disable the feature or **true** to enable the feature.

  ```bash
  $ oc patch sriovoperatorconfig default \
      --type=merge -n openshift-sriov-network-operator \
      --patch '{ "spec": { "enableInjector": <value> } }'
  ```

9.3.1.5. Disabling or enabling the SR-IOV 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 Command-line Interface (CLI), commonly known as **oc**.
- Log in as a user with **cluster-admin** privileges.
- You must have installed the SR-IOV Operator.

**Procedure**

- Set the **enableOperatorWebhook** field. Replace `<value>` with **false** to disable the feature or **true** to enable it:

  ```bash
  $ oc patch sriovoperatorconfig default --type=merge \
      -n openshift-sriov-network-operator \
      --patch '{ "spec": { "enableOperatorWebhook": <value> } }'
  ```

9.3.1.6. 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:

  ```bash
  $ 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": ""`.

Next steps

- Configuring a SR-IOV network device

9.4. CONFIGURING AN SR-IOV NETWORK DEVICE

You can configure a Single Root I/O Virtualization (SR-IOV) device in your cluster.

9.4.1. Automated discovery of SR-IOV network devices

The SR-IOV Network Operator will search 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.

One CR is created for each worker node, and shares the same name as the node. The `.status.interfaces` list provides information about the network devices on a node.

IMPORTANT

Do not modify a SriovNetworkNodeState CR. The Operator creates and manages these resources automatically.

The following is an example of a SriovNetworkNodeState CR created by the SR-IOV Network Operator:

```json
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodeState
metadata:
  name: node-25
  namespace: openshift-sriov-network-operator
ownerReferences:
  - apiVersion: sriovnetwork.openshift.io/v1
    blockOwnerDeletion: true
```
The value for the `name` parameter is the same as the name of the worker node.

The `interfaces` collection includes a list of all of the SR-IOV devices discovered by the Operator on the worker node.

9.4.2. Configuring SR-IOV network devices

NOTE
When applying the configuration specified in a SriovNetworkNodePolicy CR, the SR-IOV Operator may 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.

Prerequisites
- Install the OpenShift Command-line Interface (CLI), commonly known as oc.
- An account with cluster-admin privileges.
- You must have installed the SR-IOV Operator.

Procedure
1. Create the following SriovNetworkNodePolicy CR, and then save the YAML in the <name>-sriov-node-network.yaml file. Replace <name> with the name for this configuration.

```yaml
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: <name>
  namespace: openshift-sriov-network-operator
spec:
  resourceName: <sriov_resource_name>
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  priority: <priority>
  mtu: <mtu>
  numVfs: <num>
  nicSelector:
    vendor: "<vendor_code>"
    deviceID: "<device_id>
    pfNames: [<pf_name>, ...]
    rootDevices: [<pci_bus_id>, "...
  deviceType: <device_type>
  isRdma: false
```

1. Specify a name for the CR object.
2. Specify the namespace where the SR-IOV Operator is installed.
3. Specify the resource name of the SR-IOV device plug-in. You can create multiple SriovNetworkNodePolicy CRs for a resource name.
4. Specify the node selector to select which nodes are configured. Only SR-IOV network devices on selected nodes are configured. The SR-IOV Container Network Interface (CNI) plug-in and device plug-in are deployed only on selected nodes.
5. Optional: Specify an integer value between 0 and 99. A smaller number gets higher priority, so a priority of 10 is higher than a priority of 99. The default value is 99.
Optional: Specify a value for the maximum transmission unit (MTU) of the virtual function. The maximum MTU value can vary for different NIC models.

Specify the number of the virtual functions (VF) to create for the SR-IOV physical network device. For an Intel Network Interface Card (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.

The nicSelector mapping selects the Ethernet device for the Operator to configure. You do not need to specify values for all the parameters. It is recommended to identify the Ethernet adapter with enough precision to minimize the possibility of selecting an Ethernet 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 point to an identical device.

Optional: Specify the vendor hex code of the SR-IOV network device. The only allowed values are either 8086 or 15b3.

Optional: Specify the device hex code of SR-IOV network device. The only allowed values are 158b, 1015, 1017.

Optional: The parameter accepts an array of one or more physical function (PF) names for the Ethernet device.

The parameter accepts an array of one or more PCI bus addresses for the physical function of the Ethernet device. Provide the address in the following format: 0000:02:00.1.

Optional: Specify the driver type for the virtual functions. You can specify one of the following values: netdevice or vfio-pci. The default value is netdevice.

NOTE
For a Mellanox card to work in Data Plane Development Kit (DPDK) mode on bare metal nodes, use the netdevice driver type and set isRdma to true. For a Mellanox card to work in DPDK mode with Container-Native Virtualization (CNV), use the vfio-pci driver type and set isRdma to false.

Optional. Specify whether to enable remote direct memory access (RDMA) mode. The default value is false. Only RDMA over Converged Ethernet (RoCE) mode is supported on Mellanox Ethernet adapters.

NOTE
If isRDMA flag 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.

2. Create the SriovNetworkNodePolicy CR. Replace <name> with the name for this configuration.

   $ oc create -f <name>-sriov-node-network.yaml

   After applying the configuration update, all the Pods in sriov-network-operator namespace transition 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}'`

9.4.3. 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
```
9.5. CONFIGURING A SR-IOV NETWORK ATTACHMENT

You can configure a network attachment for an Single Root I/O Virtualization (SR-IOV) device in the cluster.

9.5.1. Configuring SR-IOV additional network

You can configure an additional network that uses SR-IOV hardware by creating a SriovNetwork Custom Resource (CR). When you create a SriovNetwork CR, the SR-IOV Operator automatically creates a NetworkAttachmentDefinition CR.

NOTE

Do not modify or delete a SriovNetwork Custom Resource (CR) if it is attached to any Pods in the running state.

Prerequisites

- Install the OpenShift Command-line Interface (CLI), commonly known as oc.
- Log in as a user with cluster-admin privileges.

Procedure

1. Create the following SriovNetwork CR, and then save the YAML in the <name>-sriov-network.yaml file. Replace <name> with a name for this additional network.

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork

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
```

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
```

Next steps

- Configuring an SR-IOV network attachment
metadata:
  name: <name>  
namespace: openshift-sriov-network-operator  
spec:
  networkNamespace: <target_namespace>  
ipam: <ipam>  
vlan: <vlan>  
resourceName: <sriov_resource_name>  
linkState: <link_state>  
maxTxRate: <max_tx_rate>  
minTxRate: <min_rx_rate>  
vlanQoS: <vlan_qos>  
  spoofChk: "<spoof_check>"  
  trust: "<trust_vf>"  
capabilities: <capabilities>

Replace `<name>` with a name for the CR. The Operator will create a NetworkAttachmentDefinition CR with same name.

Specify the namespace where the SR-IOV Operator is installed.

Optional: Replace `<target_namespace>` with the namespace where the NetworkAttachmentDefinition CR will be created. The default value is `openshift-sriov-network-operator`.

Optional: Replace `<ipam>` a configuration object for the ipam CNI plug-in as a YAML block scalar. The plug-in manages IP address assignment for the attachment definition.

Optional: Replace `<vlan>` with a Virtual LAN (VLAN) ID for the additional network. The integer value must be from 0 to 4095. The default value is 0.

Replace `<sriov_resource_name>` with the value for the `.spec.resourceName` parameter from the SrovNetworkNodePolicy CR that defines the SR-IOV hardware for this additional network.

Optional: Replace `<link_state>` with the link state of Virtual Function (VF). Allowed value are `enable`, `disable` and `auto`.

Optional: Replace `<max_tx_rate>` with a maximum transmission rate, in Mbps, for the VF.

Optional: Replace `<min_rx_rate>` with a minimum transmission rate, in Mbps, for the VF. This value should always be less than or equal to Maximum transmission rate.

NOTE

Intel NICs do not support the `minTxRate` parameter. For more information, see `BZ#1772847`.

Optional: Replace `<vlan_qos>` with an IEEE 802.1p priority level for the VF. The default value is 0.

Optional: Replace `<spoof_check>` with 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 CR will be rejected by the SR-IOV Network Operator.

 Optional: Replace `<trust_vf>` with the trust 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 CR will be rejected by the SR-IOV Network Operator.

 Optional: Replace `<capabilities>` with the capabilities to configure for this network. You can specify "{ "ips": true }" to enable IP address support or "{ "mac": true }" to enable MAC address support.

 2. Create the CR by running the following command:

     $ oc create -f <filename>  

     Replace `<filename>` with the name of the file you created in the previous step.

 3. Optional: Confirm that the NetworkAttachmentDefinition CR associated with the SriovNetwork CR that you created in the previous step exists by running the following command. Replace `<namespace>` with the namespace you specified in the SriovNetwork CR.

     oc get net-attach-def -n <namespace>

9.5.1.1. Configuration for ipam CNI plug-in

The IP address management (IPAM) Container Network Interface (CNI) plug-in manages IP address assignment for other CNI plug-ins. You can configure ipam for either static IP address assignment or dynamic IP address assignment by using DHCP. The DHCP server you specify must be reachable from the additional network.

The following JSON configuration object describes the parameters that you can set.

 IMPORTANT 
 If you set the type parameter to the DHCP value, you cannot set any other parameters.

 ipam CNI plug-in JSON configuration object

```json
{
  "ipam": {
    "type": "<type>",  
    "addresses": [
      {
        "address": "<address>",  
        "gateway": "<gateway>"
      }
    ]
  }
}
```
Specify **static** to configure the plug-in to manage IP address assignment. Specify **DHCP** to allow a DHCP server to manage IP address assignment. You cannot specify any additional parameters if you specify a value of **DHCP**.

An array describing IP addresses to assign to the virtual interface. Both IPv4 and IPv6 IP addresses are supported.

A block of IP addresses that you specify in CIDR format to assign to Pods on a worker node, such as `10.1.1.0/24`.

The default gateway to route egress network traffic to.

An array describing routes to configure inside the Pod.

The IP address range in CIDR format.

The gateway where network traffic is routed.

Optional: DNS configuration.

An array of one or more IP addresses for to send DNS queries to.

The default domain to append to a host name. For example, if the domain is set to `example.com`, a DNS lookup query for `example-host` is rewritten as `example-host.example.com`.

An array of domain names to append to an unqualified host name, such as `example-host`, during a DNS lookup query.

### 9.5.1.1.1. Static IP address assignment configuration example

You can configure ipam for static IP address assignment:

```json
{
  "ipam": {
    "type": "static",
    "addresses": [
      { "address": "191.168.1.1/24"
    ]
  }
}
```
9.5.1.1.2. Dynamic IP address assignment configuration example

You can configure ipam for DHCP:

```yaml
{
  "ipam": {
    "type": "DHCP"
  }
}
```

9.5.1.2. Configuring static MAC and IP addresses on additional SR-IOV networks

You can configure static MAC and IP addresses on an SR-IOV network by specifying Container Network Interface (CNI) runtimeConfig data in a pod annotation.

**Prerequisites**

- Install the OpenShift Command-line Interface (CLI), commonly known as `oc`.
- Log in as a user with `cluster-admin` privileges when creating the SriovNetwork CR.

**Procedure**

1. Create the following SriovNetwork CR, and then save the YAML in the `<name>-sriov-network.yaml` file. Replace `<name>` with a name for this additional network.

```yaml
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: <name>  
  namespace: sriov-network-operator
spec:
  networkNamespace: <target_namespace>
  ipam: '{ "type": "static" }'
  capabilities: '{ "mac": true, "ips": true }'
  resourceName: <sriov_resource_name>

```

Replace `<name>` with a name for the CR. The SR-IOV Network Operator creates a NetworkAttachmentDefinition CR with same name.

Specify the namespace where the SR-IOV Network Operator is installed.

Replace `<target_namespace>` with the namespace where the NetworkAttachmentDefinition CR is created.

Specify static type for the ipam CNI plug-in as a YAML block scalar.

Specify `mac` and `ips` capabilities to `true`. 

---

OpenShift Container Platform 4.4 Networking
Replace `<sriov_resource_name>` with the value for the `spec.resourceName` parameter from the SriovNetworkNodePolicy CR that defines the SR-IOV hardware for this

2. Create the CR by running the following command:

```
$ oc create -f <filename>
```

Replace `<filename>` with the name of the file you created in the previous step.

3. Optional: Confirm that the NetworkAttachmentDefinition CR associated with the SriovNetwork CR that you created in the previous step exists by running the following command. Replace `<namespace>` with the namespace you specified in the SriovNetwork CR.

```
$ oc get net-attach-def -n <namespace>
```

**NOTE**

Do not modify or delete a SriovNetwork Custom Resource (CR) if it is attached to any Pods in the `running` state.

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.

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: sample-pod
annotations:
  k8s.v1.cni.cncf.io/networks: |
  "name": "<name>",
  "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"]
```

Replace `<name>` with then name of the SR-IOV network attachment definition CR.

Specify the `mac` address for the SR-IOV device which is allocated from the resource type defined in the SR-IOV network attachment definition CR.

Specify the IPv4 and/or IPv6 addresses for the SR-IOV device which is allocated from the resource type defined in the SR-IOV network attachment definition CR.

2. Create the sample SR-IOV pod by running the following command:
$ oc create -f <filename>  

Replace `<filename>` with the name of the file you created in the previous step.

3. Optional: Confirm that `mac` and `ips` addresses are applied to the SR-IOV device by running the following command. Replace `<namespace>` with the namespace you specified in the `SriovNetwork CR`.

$ oc exec sample-pod -n <namespace> -- ip addr show

Next steps

- Adding a Pod to an SR-IOV additional network

9.6. 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.

9.6.1. 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.

**NOTE**

The Network Resources Injector will inject the `resource` parameter into the Pod CR automatically if a `NetworkAttachmentDefinition CR` associated with the SR-IOV CNI plug-in is specified.

Prerequisites

- The Pod must be in the same namespace as the additional network.
- Install the OpenShift Command-line Interface (CLI), commonly known as `oc`.
- You must log in to the cluster.
- You must have the SR-IOV Operator installed and a `SriovNetwork CR` defined.

Procedure

To add a Pod with additional networks, complete the following steps:

1. Create the Pod resource definition and add the `k8s.v1.cni.cncf.io/networks` parameter to the Pod `metadata` mapping. The `k8s.v1.cni.cncf.io/networks` accepts a comma separated string of one or more `NetworkAttachmentDefinition CR` names:

```yaml
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: <network>[,<network>,...]
```
Replace `<network>` with the name of the additional network to associate with the Pod. To specify more than one additional network, separate each network with a comma. Do not

In the following example, two additional networks are attached to the Pod:

```
apiVersion: v1
crdd: Pod
metadata:
  name: example-pod
  annotations:
    k8s.v1.cni.cncf.io/networks: net1,net2
spec:
  containers:
  - name: example-pod
    command: ['/bin/bash', '-c', 'sleep 2000000000000']
  image: centos/tools
```

2. Create the Pod by running the following command:

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

3. Optional: Confirm that the annotation exists in the Pod CR by running the following command. Replace `<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
crdd: Pod
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: macvlan-bridge
    k8s.v1.cni.cncf.io/networks-status:
      -
  name: example-pod
  namespace: default
```
The `k8s.v1.cni.cncf.io/networks-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.

9.6.2. 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

- Install the OpenShift Command-line Interface (CLI), commonly known as `oc`.
- Enable a LatencySensitive profile and configure the CPU Manager policy to `static`.

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 a SR-IOV pod spec:

   ```yaml
   apiVersion: v1
   kind: Pod
   metadata:
     name: sample-pod
   annotations:
     k8s.v1.cni.cncf.io/networks: <name>  
   spec:
     containers:
     - name: sample-container
       image: <image>  
       command: ["sleep", "infinity"]
       resources:
         limits:
           memory: "1Gi"
           cpu: "2"
         requests:
           memory: "1Gi"
           cpu: "2"
   ``

   Replace `<name>` with the name of the SR-IOV network attachment definition CR.

   Replace `<image>` with the name of the `sample-pod` image.

   To create the SR-IOV pod with guaranteed QoS, set `memory limits` equal to `memory requests`.

   To create the SR-IOV pod with guaranteed QoS, set `cpu limits` equals to `cpu requests`.

   ```yaml
   spec:
     ...
   status:
     ...
   ```
2. Create the sample SR-IOV pod by running the following command:

   ```shell
   $ oc create -f <filename>
   ```

   Replace `<filename>` with the name of the file you created in the previous step.

3. Optional: Confirm that the `sample-pod` is configured with guaranteed QoS.

   ```shell
   oc describe pod sample-pod
   ```

4. Optional: Confirm that the `sample-pod` is allocated with exclusive CPUs.

   ```shell
   oc exec sample-pod -- cat /sys/fs/cgroup/cpuset/cpuset.cpus
   ```

5. Optional: Confirm that the SR-IOV device and CPUs that are allocated for the `sample-pod` are on the same NUMA node.

   ```shell
   oc exec sample-pod -- cat /sys/fs/cgroup/cpuset/cpuset.cpus
   ```

9.7. USING HIGH PERFORMANCE MULTICAST

   You can use multicast on your Single Root I/O Virtualization (SR-IOV) hardware network.

9.7.1. Configuring high performance multicast

   The OpenShift SDN default Container Network Interface (CNI) network provider 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.

9.7.2. Using an SR-IOV interface for multicast

   The follow procedure creates an example SR-IOV interface for multicast.

   **Prerequisites**
   
   - Install the OpenShift Command-line Interface (CLI), commonly known as `oc`.
   
   - You must log in to the cluster with a user that has the `cluster-admin` role.

   **Procedure**
   
   1. Create a SriovNetworkNodePolicy custom resource (CR):
Create a SriovNetwork CR:

```yaml
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 CR:

```yaml
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: net-example
namespace: openshift-sriov-network-operator
spec:
  networkNamespace: default
  ipam:
    | 1
    | {  
    |   "type": "host-local",  
    |   "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
```

1. 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:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: testpmd
namespace: default
annotations:
  k8s.v1.cni.cncf.io/networks: nic1
spec:
  containers:
    - name: example
```
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.

### 9.8. USING VIRTUAL FUNCTIONS (VFS) WITH DPDK AND RDMA MODES

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).

#### 9.8.1. Examples of using virtual functions in DPDK and RDMA modes

**IMPORTANT**

The Data Plane Development Kit (DPDK) 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 [https://access.redhat.com/support/offerings/techpreview/](https://access.redhat.com/support/offerings/techpreview/).

**IMPORTANT**

Remote Direct Memory Access (RDMA) 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 [https://access.redhat.com/support/offerings/techpreview/](https://access.redhat.com/support/offerings/techpreview/).

**Prerequisites**

- Install the OpenShift Command-line Interface (CLI), commonly known as `oc`.
- Log in as a user with `cluster-admin` privileges.
- You must have installed the SR-IOV Operator.

#### 9.8.1.1. Example use of virtual function (VF) in DPDK mode with Intel NICs

**Procedure**

```yaml
image: rhel7:latest
securityContext:
capabilities:
  add: ["NET_ADMIN"]
command: ["sleep", "infinity"]
```
1. Create the following SriovNetworkNodePolicy CR, and then save the YAML in the `intel-dpdk-node-policy.yaml` file.

```yaml
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
```

Specify the driver type for the virtual functions to `vfio-pci`.

**NOTE**

Please refer to the Configuring SR-IOV network devices section for a detailed explanation on each option in SriovNetworkNodePolicy.

+ When applying the configuration specified in a SriovNetworkNodePolicy CR, 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 CR by running the following command:

```bash
$ oc create -f intel-dpdk-node-policy.yaml
```

3. Create the following SriovNetwork CR, and then save the YAML in the `intel-dpdk-network.yaml` file.

```yaml
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: intel-dpdk-network
  namespace: openshift-sriov-network-operator
spec:
  networkNamespace: <target_namespace>
```
Specify an empty object "{}" for the ipam CNI plug-in. DPDK works in userspace mode and does not require an IP address.

NOTE
Please refer to the Configuring SR-IOV additional network section for a detailed explanation on each option in SriovNetwork.

4. Create the SriovNetworkNodePolicy CR by running the following command:

```bash
$ 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.

```yaml
apiVersion: v1
crde.k8s.io/v1
kind: Pod
metadata:
  name: dpdk-app
  namespace: <target_namespace>
  annotations:
    k8s.v1.cni.cncf.io/networks: intel-dpdk-network
spec:
  containers:
  - name: testpmd
    image: <DPDK_image>
    securityContext:
      capabilities:
        add: ["IPC_LOCK"]
    volumeMounts:
    - mountPath: /dev/hugepages
      name: hugepage
    resources:
      limits:
        openshift.io/intelnics: "1"
        memory: "1Gi"
        cpu: "4"
        hugepages-1Gi: "4Gi"
      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 CR `intel-dpdk-network` is created. If you would like to create the Pod in a different namespace, change

2. Specify the DPDK image which includes your application and the DPDK library used by application.

3. Specify the **IPC_LOCK** capability which is required by the application to allocate hugepage memory inside container.

4. Mount a hugepage volume to the DPDK Pod under `/dev/hugepages`. 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 `Injector` 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
```

9.8.1.2. Example use of a virtual function in DPDK mode with Mellanox NICs

**Procedure**

1. Create the following SriovNetworkNodePolicy CR, and then save the YAML in the `mlx-dpdk-node-policy.yaml` file.

```yaml
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: mlx-dpdk-node-policy
  namespace: openshift-sriov-network-operator
spec:
  resourceName: mlnics
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  priority: <priority>
  numVFs: <num>
  nicSelector:
    vendor: "15b3"
    deviceID: "1015"
    pfNames: ["<pf_name>", ...]
```

---

1. Specify the same **target_namespace** where the SriovNetwork CR `intel-dpdk-network` is created. If you would like to create the Pod in a different namespace, change

2. Specify the DPDK image which includes your application and the DPDK library used by application.

3. Specify the **IPC_LOCK** capability which is required by the application to allocate hugepage memory inside container.

4. Mount a hugepage volume to the DPDK Pod under `/dev/hugepages`. 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 `Injector` 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
```

9.8.1.2. Example use of a virtual function in DPDK mode with Mellanox NICs

**Procedure**

1. Create the following SriovNetworkNodePolicy CR, and then save the YAML in the `mlx-dpdk-node-policy.yaml` file.

```yaml
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: mlx-dpdk-node-policy
  namespace: openshift-sriov-network-operator
spec:
  resourceName: mlnics
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  priority: <priority>
  numVFs: <num>
  nicSelector:
    vendor: "15b3"
    deviceID: "1015"
    pfNames: ["<pf_name>", ...]
```
Specify the device hex code of the SR-IOV network device. The only allowed values for Mellanox cards are 1015, 1017.

Specify the driver type for the virtual functions to netdevice. Mellanox SR-IOV VF can work in DPDK mode without using the vfio-pci device type. VF device appears as a kernel network interface inside a container.

Enable RDMA mode. This is required by Mellanox cards to work in DPDK mode.

**NOTE**

Please refer to Configuring SR-IOV network devices section for detailed explanation on each option in SriovNetworkNodePolicy.

+ When applying the configuration specified in a SriovNetworkNodePolicy CR, 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 CR by running the following command:

   $ oc create -f mlx-dpdk-node-policy.yaml

3. Create the following SriovNetwork CR, and then save the YAML in the mlx-dpdk-network.yaml file.

   ```yaml
   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: mlxnic
   ```

   Specify a configuration object for the ipam CNI plug-in as a YAML block scalar. The plug-in manages IP address assignment for the attachment definition.
NOTE

Please refer to Configuring SR-IOV additional network section for detailed explanation on each option in SriovNetwork.

4. Create the SriovNetworkNodePolicy CR by running the following command:

   $ oc create -f mlx-dpdk-network.yaml

5. Create the following Pod spec, and then save the YAML in the mlx-dpdk-pod.yaml file.

   ```yaml
   apiVersion: v1
   kind: Pod
   metadata:
     name: dpdk-app
     namespace: <target_namespace>  
   annotations:
     k8s.v1.cni.cncf.io/networks: mlx-dpdk-network
   spec:
     containers:
     - name: testpmd
       image: <DPDK_image>  
       securityContext:
         capabilities:
           add:
             - IPC_LOCK
       volumeMounts:
       - mountPath: /dev/hugepages
         name: hugepage
       resources:
         limits:
           openshift.io/mlxnics: "1"  
           memory: "1Gi"
           cpu: "4"  
           hugepages-1Gi: "4Gi"
         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 CR mlx-dpdk-network is created. If you would like to create the Pod in a different namespace, change target_namespace in both Pod spec and SriovNetwork CR.

2. Specify the DPDK image which includes your application and the DPDK library used by application.

3. Specify the IPC_LOCK capability which is required by the application to allocate hugepage memory inside the container.
4. Mount the hugepage volume to the DPDK Pod under `/dev/hugepages`. 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 the DPDK Pod. This resource request and limit, if not explicitly specified, will be automatically added by SR-IOV network resource injector. The SR-IOV network resource injector is an admission controller component managed by SR-IOV Operator. It is enabled by default and can be disabled by setting the `Injector` option to `false` in the default `SriovOperatorConfig` CR.

6. Specify the number of CPUs. The DPDK Pod usually requires exclusive CPUs be allocated from 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 DPDK Pod. Configure `2Mi` and `1Gi` hugepages separately. Configuring `1Gi` hugepage requires adding kernel arguments to Nodes.

6. Create the DPDK Pod by running the following command:

```
$ oc create -f mlx-dpdk-pod.yaml
```

9.8.1.3. Example of a virtual function in RDMA mode with Mellanox NICs

RDMA over Converged Ethernet (RoCE) is the only supported mode when using RDMA on OpenShift Container Platform.

Procedure

1. Create the following SriovNetworkNodePolicy CR, and then save the YAML in the `mlx-rdma-node-policy.yaml` file.

```yaml
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: mlx-rdma-node-policy
  namespace: openshift-sriov-network-operator
spec:
  resourceName: mlxvnics
  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 SR-IOV network device. The only allowed values for Mellanox cards are `1015, 1017`.

2. Specify the driver type for the virtual functions to `netdevice`.
3. Enable RDMA mode.

**NOTE**

Please refer to the Configuring SR-IOV network devices section for a detailed explanation on each option in SriovNetworkNodePolicy.

+ When applying the configuration specified in a SriovNetworkNodePolicy CR, 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 CR by running the following command:

   ```bash
   $ oc create -f mlx-rdma-node-policy.yaml
   ```

3. Create the following SriovNetwork CR, and then save the YAML in the mlx-rdma-network.yaml file.

   ```yaml
   apiVersion: sriovnetwork.openshift.io/v1
   kind: SriovNetwork
   metadata:
     name: mlx-rdma-network
     namespace: openshift-sriov-network-operator
   spec:
     networkNamespace: <target_namespace>
     ipam: |
       ...  
       vlan: <vlan>
     resourceName: mlnics
   ```

   1. Specify a configuration object for the ipam CNI plug-in as a YAML block scalar. The plug-in manages IP address assignment for the attachment definition.

   **NOTE**

   Please refer to Configuring SR-IOV additional network section for detailed explanation on each option in SriovNetwork.

4. Create the SriovNetworkNodePolicy CR by running the following command:

   ```bash
   $ 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.

   ```yaml
   apiVersion: v1
   kind: Pod
   ```
Specify the same target_namespace where SriovNetwork CR 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 SriovNetowrk CR.

Specify the RDMA image which includes your application and RDMA library used by application.

Specify the IPC_LOCK capability which is required by the application to allocate hugepage memory inside the container.

Mount the hugepage volume to RDMA Pod under /dev/hugepages. The hugepage volume is backed by the emptyDir volume type with the medium being HugePages.

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.

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:

```bash
$ oc create -f mlx-rdma-pod.yaml
```
CHAPTER 10. OPENShift SDN DEFAULT CNI NETWORK PROVIDER

10.1. ABOUT THE OPENShift SDN DEFAULT CNI NETWORK PROVIDER

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. This Pod network is established and maintained by the OpenShift SDN, which configures an overlay network using Open vSwitch (OVS).

OpenShift SDN provides three SDN modes for configuring the Pod network:

- The **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 4.4.

- The **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.

- The **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 the subnet mode.

10.2. CONFIGURING EGRESS IPS FOR A PROJECT

As a cluster administrator, you can configure the OpenShift SDN default Container Network Interface (CNI) network provider to assign one or more egress IP addresses to a project.

10.2.1. Egress IP address assignment for project egress traffic

By configuring an egress IP address for a project, all outgoing external connections from the specified project will share the same, fixed source IP address. External resources can recognize traffic from a particular project based on the egress IP address. An egress IP address assigned to a project is different from the egress router, which is used to send traffic to specific destinations.

Egress IP addresses are implemented as additional IP addresses on the primary network interface of the node and must be in the same subnet as the node’s primary IP address.

**IMPORTANT**

Egress IP addresses must not be configured in any Linux network configuration files, such as `ifcfg-eth0`.

Allowing additional IP addresses on the primary network interface might require extra configuration when using some cloud or VM solutions.

You can assign egress IP addresses to namespaces by setting the `egressIPs` parameter of the `NetNamespace` resource. After an egress IP is associated with a project, OpenShift SDN allows you to assign egress IPs 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 resource, 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.

IMPORTANT

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.

10.2.1.1. 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.

- Only a single egress IP address per namespace is supported when using the automatic assignment mode.

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.

10.2.1.2. Considerations when using manually assigned egress IP addresses

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.

When a namespace has multiple egress IP addresses, if the node hosting the first egress IP address is unreachable, OpenShift Container Platform will automatically switch to using the next available egress IP address until the first egress IP address is reachable again.

This approach is recommended for clusters installed in public cloud environments, where there can be limitations on associating additional IP addresses with nodes.

10.2.2. 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

- Install the OpenShift Command-line Interface (CLI), commonly known as `oc`.
- Access to the cluster as a user with the `cluster-admin` role.

Procedure

1. Update the `NetNamespace` resource with the egress IP address using the following JSON:

   ```bash
   $ oc patch netnamespace <project_name> --type=merge -p \\
   '{
   "egressIPs": [
   "<ip_address>"
   ]
   }
   '
   
   Specify the name of the project.

   Specify a single egress IP address. Using multiple IP addresses is not supported.

   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:

   ```bash
   $ oc patch netnamespace project1 --type=merge -p \\
   '{"egressIPs": ["192.168.1.100"]}'
   $ oc patch netnamespace project2 --type=merge -p \\
   '{"egressIPs": ["192.168.1.101"]}'
   
   2. Indicate which nodes can host egress IP addresses by setting the `egressCIDRs` parameter for each host using the following JSON:

   ```bash
   $ oc patch hostsubnet <node_name> --type=merge -p \\
   '{
   "egressCIDRs": [
   "<ip_address_range_1>", "<ip_address_range_2>", 
   ]
   }'
   
   Specify a node name.

   Specify one or more IP address ranges in CIDR format.

   For example, to set `node1` and `node2` to host egress IP addresses in the range 192.168.1.0 to 192.168.1.255:

   ```bash
   $ oc patch hostsubnet node1 --type=merge -p \\
   '{"egressCIDRs": ["192.168.1.0/24"]}'
   $ oc patch hostsubnet node2 --type=merge -p \\
   '{"egressCIDRs": ["192.168.1.255/24"]}'
   ```
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.

10.2.3. 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**

- Install the OpenShift Command-line Interface (CLI), commonly known as `oc`.
- Access to the cluster as a user with the `cluster-admin` role.

**Procedure**

1. Update the `NetNamespace` resource by specifying the following JSON object with the desired IP addresses:

   ```bash
   $ oc patch netnamespace <project> --type=merge -p \1
   '{
      "egressIPs": [2
        "<ip_address>"
      ]
   }
   
   1 Specify the name of the project.
   2 Specify one or more egress IP addresses. The `egressIPs` parameter is an array.

   For example, to assign the `project1` project to an IP address of `192.168.1.100`:

   ```bash
   $ oc patch netnamespace project1 --type=merge -p '{"egressIPs": ["192.168.1.100"]}'
   
   You can set `egressIPs` to two or more IP addresses on different nodes to provide high availability. If multiple egress IP addresses are set, pods use the first IP in the list for egress, but if the node hosting that IP address fails, pods switch to using the next IP in the list after a short delay.

2. Manually assign the egress IP to the node hosts. Set the `egressIPs` parameter on the `HostSubnet` object on the node host. Using the following JSON, include as many IPs as you want to assign to that node host:

   ```bash
   $ oc patch hostsubnet <node_name> --type=merge -p \1
   '{
      "egressIPs": [2
        "<ip_address_1>",
        "<ip_address_N>"
      ]
   }
   
   1 Specify the name of the node host.
   2 Specify one or more egress IP addresses on different nodes.
Specify the name of the project.

Specify one or more egress IP addresses. The `egressIPs` field is an 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 (using NAT) to that IP address.

# 10.3. CONFIGURING AN EGRESS FIREWALL TO CONTROL ACCESS TO EXTERNAL IP ADDRESSES

As a cluster administrator, you can create an egress firewall for a project that will restrict egress traffic leaving your OpenShift Container Platform cluster.

## 10.3.1. 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.

You configure an egress firewall policy by creating an EgressNetworkPolicy Custom Resource (CR) object and specifying an IP address range in CIDR format or by specifying a DNS name. 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.

**IMPORTANT**

You must have OpenShift SDN configured to use either the network policy or multitenant modes to configure egress firewall policy.

If you use network policy mode, egress policy is compatible with only one policy per namespace and will not work with projects that share a network, such as global projects.

**CAUTION**

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 network policy rules by creating a route that points to a forbidden destination.
10.3.1.1. Limitations of an egress firewall

An egress firewall has the following limitations:

- No project can have more than one EgressNetworkPolicy object.
- The default project cannot use egress network policy.
- When using the OpenShift SDN default Container Network Interface (CNI) network provider 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 broken egress network policy for the project, and may cause all external network traffic to be dropped.

10.3.1.2. Matching order for egress network policy rules

The egress network 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.

10.3.1.3. 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 the TTL (time to live) value of the domain returned by the local non-authoritative servers.
- 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 host name 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**

The egress firewall always allows Pods access to the external interface of the node that the Pod is on for DNS resolution.

If you use domain names in your egress firewall policy and your DNS resolution is not handled by a DNS server on the local node, then you must add egress firewall rules that allow access to your DNS server’s IP addresses. If you are using domain names in your Pods.

10.3.2. EgressNetworkPolicy custom resource (CR) object
The following YAML describes an EgressNetworkPolicy CR object:

```
kind: EgressNetworkPolicy
apiVersion: v1
metadata:
  name: <name> 1
spec:
  egress: 2
...
```

1 Specify a **name** for your egress firewall policy.
2 Specify a collection of one or more egress network policy rules as described in the following section.

### 10.3.2.1. EgressNetworkPolicy rules

The following YAML describes an egress firewall rule object. The **egress** key expects an array of one or more objects.

```
egress:
- type: <type> 1
to: 2
cidrSelector: <cidr> 3
dnsName: <dns-name> 4
```

1 Specify the type of rule. The value must be either **Allow** or **Deny**.
2 Specify a value for either the **cidrSelector** key or the **dnsName** key for the rule. You cannot use both keys in a rule.
3 Specify an IP address range in CIDR format.
4 Specify a domain name.

### 10.3.2.2. Example EgressNetworkPolicy CR object

The following example defines several egress firewall policy rules:

```
kind: EgressNetworkPolicy
apiVersion: v1
metadata:
  name: default-rules 1
spec:
  egress: 2
- 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. The name for the policy object.
2. A collection of egress firewall policy rule objects.

### 10.3.3. 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 default Container Network Interface (CNI) network provider plug-in.
- Install the OpenShift Command-line Interface (CLI), commonly known as `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:

   ```bash
   $ oc create -f <policy-name>.yaml -n <project>
   ``

   In the following example, a new EgressNetworkPolicy object is created in a project named `project1`:

   ```bash
   $ oc create -f default-rules.yaml -n project1
   egressnetworkpolicy.network.openshift.io/default-rules created
   ```

3. Optional: Save the `<policy-name>.yaml` so that you can make changes later.

### 10.4. EDITING AN EGRESS FIREWALL FOR A PROJECT

As a cluster administrator, you can modify network traffic rules for an existing egress firewall.

#### 10.4.1. 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 plug-in.
- Install the OpenShift Command-line Interface (CLI), commonly known as oc.
- You must log in to the cluster as a cluster administrator.

Procedure

To edit an existing egress network policy object for a project, complete the following steps:

1. Find the name of the EgressNetworkPolicy object for the project. Replace <project> with the name of the project.

   ```bash
   $ 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.

   ```bash
   $ 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.

3. Enter the following command to replace the EgressNetworkPolicy object. Replace <filename> with the name of the file containing the updated EgressNetworkPolicy object.

   ```bash
   $ oc replace -f <filename>.yaml
   ```

10.4.2. EgressNetworkPolicy custom resource (CR) object

The following YAML describes an EgressNetworkPolicy CR object:

```yaml
kind: EgressNetworkPolicy
apiVersion: v1
metadata:
  name: <name>  
spec:
  egress: ...
```

1. Specify a name for your egress firewall policy.
2. Specify a collection of one or more egress network policy rules as described in the following section.
10.4.2.1. EgressNetworkPolicy rules

The following YAML describes an egress firewall rule object. The egress key expects an array of one or more objects.

```yaml
egress:
- type: <type>  
to:
  cidrSelector: <cidr>  
dnsName: <dns-name>
```

1. Specify the type of rule. The value must be either Allow or Deny.
2. Specify a value for either the cidrSelector key or the dnsName key for the rule. You cannot use both keys in a rule.
3. Specify an IP address range in CIDR format.
4. Specify a domain name.

10.4.2.2. Example EgressNetworkPolicy CR object

The following example defines several egress firewall policy rules:

```yaml
kind: EgressNetworkPolicy
apiVersion: v1
metadata:
  name: default-rules
spec:
  egress:
  - 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. The name for the policy object.
2. A collection of egress firewall policy rule objects.

10.5. 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.

10.5.1. 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 plug-in.
- Install the OpenShift Command-line Interface (CLI), commonly known as oc.
- You must log in to the cluster as a cluster administrator.

Procedure

To remove an egress network policy object for a project, complete the following steps:

1. Find the name of the EgressNetworkPolicy object for the project. Replace `<project>` with the name of the project.
   
   ```bash
   $ 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.
   
   ```bash
   $ oc delete -n <project> egressnetworkpolicy <name>
   ```

10.6. ENABLING MULTICAST FOR A PROJECT

10.6.1. 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.

Multicast traffic between OpenShift Container Platform Pods is disabled by default. If you are using the OpenShift SDN default Container Network Interface (CNI) network provider, you can enable multicast on a per-project basis.

When using the OpenShift SDN network plug-in 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 plug-in 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.

10.6.2. Enabling multicast between Pods
You can enable multicast between Pods for your project.

Prerequisites

- Install the OpenShift Command-line Interface (CLI), commonly known as `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:

  ```
  $ oc annotate netnamespace <namespace> \
  netnamespace.network.openshift.io/multicast-enabled=true
  ```

  The `namespace` for the project you want to enable multicast for.

10.7. DISABLING MULTICAST FOR A PROJECT

10.7.1. Disabling multicast between Pods

You can disable multicast between Pods for your project.

Prerequisites

- Install the OpenShift Command-line Interface (CLI), commonly known as `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> \
  netnamespace.network.openshift.io/multicast-enabled=false
  ```

  The `namespace` for the project you want to disable multicast for.

10.8. CONFIGURING NETWORK ISOLATION USING OPENSHIFT SDN

When your cluster is configured to use the multitenant isolation mode for the OpenShift SDN CNI plug-in, 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.

**Prerequisites**

- You must have a cluster configured to use the OpenShift SDN Container Network Interface (CNI) plug-in in multitenant isolation mode.

### 10.8.1. Joining projects

You can join two or more projects to allow network traffic between Pods and services in different projects.

**Prerequisites**

- Install the OpenShift Command-line Interface (CLI), commonly known as `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:

   ```shell
   $ 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:

   ```shell
   $ oc get netnamespaces
   ```

   Projects in the same pod-network have the same network ID in the `NETID` column.

### 10.8.2. 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 Command-line Interface (CLI), commonly known as `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:

  ```shell
  $ 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.
10.8.3. Disabling network isolation for a project

You can disable network isolation for a project.

Prerequisites

- Install the OpenShift Command-line Interface (CLI), commonly known as `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.

10.9. 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.

10.9.1. 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.

10.9.2. Modifying the kube-proxy configuration

You can modify the Kubernetes network proxy configuration for your cluster.

Prerequisites

- Install the OpenShift Command-line Interface (CLI), commonly known as `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:

```yaml
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. Run the following command to confirm the configuration update:

   ```bash
   $ oc get networks.operator.openshift.io -o yaml
   ```

   The command returns output similar to the following example:

   ```yaml
   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: Run the following command to confirm that the Cluster Network Operator accepted the configuration change:

   ```bash
   $ oc get clusteroperator network
   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.
10.9.3. kube-proxy configuration parameters

You can modify the following `kubeProxyConfig` parameters:

**Table 10.1. Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Values</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>iptablesSyncPeriod</code></td>
<td>The refresh period for <code>iptables</code> rules.</td>
<td>A time interval, such as <strong>30s</strong> or <strong>2m</strong>. Valid suffixes include <strong>s</strong>, <strong>m</strong>, and <strong>h</strong> and are described in the Go time package documentation.</td>
<td><strong>30s</strong></td>
</tr>
<tr>
<td><code>proxyArguments.iptables-min-sync-period</code></td>
<td>The minimum duration before refreshing <code>iptables</code> rules. This parameter ensures that the refresh does not happen too frequently.</td>
<td>A time interval, such as <strong>30s</strong> or <strong>2m</strong>. Valid suffixes include <strong>s</strong>, <strong>m</strong>, and <strong>h</strong> and are described in the Go time package</td>
<td><strong>30s</strong></td>
</tr>
</tbody>
</table>
CHAPTER 11. OVN-KUBERNETES DEFAULT CNI NETWORK PROVIDER

11.1. ABOUT THE OVN-KUBERNETES DEFAULT CONTAINER NETWORK INTERFACE (CNI) NETWORK PROVIDER

IMPORTANT

The Open Virtual Networking (OVN) Kubernetes network plug-in 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 the OVN Technology Preview, see https://access.redhat.com/articles/4380121.

The OpenShift Container Platform cluster uses a virtualized network for Pod and Service networks. The OVN-Kubernetes Container Network Interface (CNI) plug-in is a network provider for the default cluster network.

11.1.1. OVN-Kubernetes features

The OVN-Kubernetes default Container Network Interface (CNI) network provider implements the following features:

- 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.

11.1.2. Supported default CNI network provider feature matrix

OpenShift Container Platform offers two supported choices, OpenShift SDN and OVN-Kubernetes, for the default Container Network Interface (CNI) network provider. The following table summarizes the current feature support for both network providers:

<table>
<thead>
<tr>
<th>Feature</th>
<th>OVN-Kubernetes</th>
<th>OpenShift SDN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Egress IPs</td>
<td>Not supported</td>
<td>Supported</td>
</tr>
<tr>
<td>Egress firewall</td>
<td>Not supported</td>
<td>Supported</td>
</tr>
<tr>
<td>Egress router</td>
<td>Not supported</td>
<td>Supported</td>
</tr>
</tbody>
</table>
### Feature

<table>
<thead>
<tr>
<th>Feature</th>
<th>OVN-Kubernetes (a)</th>
<th>OpenShift SDN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kubernetes network policy</td>
<td>Supported</td>
<td>Partially supported (c)</td>
</tr>
<tr>
<td>Multicast</td>
<td>Supported</td>
<td>Supported</td>
</tr>
</tbody>
</table>

(a) Available only as a Technology Preview feature in OpenShift Container Platform 4.4.

(b) Egress firewall is also known as egress network policy in OpenShift SDN. This is not the same as network policy egress.

(c) Does not support egress rules and some ipBlock rules.

### 11.1.3. Exposed metrics for OVN-Kubernetes

The OVN-Kubernetes default Container Network Interface (CNI) network provider exposes certain metrics for use by the Prometheus-based OpenShift Container Platform cluster monitoring stack.

**Table 11.2. Metrics exposed by OVN-Kubernetes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ovnkube_master_pod_creation_latency_seconds</code></td>
<td>The latency between when a Pod is created and when the Pod is annotated by OVN-Kubernetes. The higher the latency, the more time that elapses before a Pod is available for network connectivity.</td>
</tr>
</tbody>
</table>

### Additional resources

- Enabling multicast for a project
- Disabling multicast for a project

### 11.2. ENABLING MULTICAST FOR A PROJECT

**IMPORTANT**

The Open Virtual Networking (OVN) Kubernetes network plug-in 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 the OVN Technology Preview, see [https://access.redhat.com/articles/4380121](https://access.redhat.com/articles/4380121).

### 11.2.1. 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.

Multicast traffic between OpenShift Container Platform Pods is disabled by default. If you are using the OVN-Kubernetes default Container Network Interface (CNI) network provider, you can enable multicast on a per-project basis.

11.2.2. Enabling multicast between Pods

You can enable multicast between Pods for your project.

Prerequisites

- Install the OpenShift Command-line Interface (CLI), commonly known as `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:

  ```bash
  $ oc annotate namespace <namespace> \
  k8s.ovn.org/multicast-enabled=true
  ```

  The `namespace` for the project you want to enable multicast for.

11.3. DISABLING MULTICAST FOR A PROJECT

IMPORTANT

The Open Virtual Networking (OVN) Kubernetes network plug-in 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 the OVN Technology Preview, see https://access.redhat.com/articles/4380121.

11.3.1. Disabling multicast between Pods

You can disable multicast between Pods for your project.

Prerequisites

- Install the OpenShift Command-line Interface (CLI), commonly known as `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> \\
k8s.ovn.org/multicast-enabled-1
```

1 The `namespace` for the project you want to disable multicast for.
CHAPTER 12. CONFIGURING ROUTES

12.1. ROUTE CONFIGURATION

12.1.1. 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>
   ```

   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
   ```

12.1.2. Enabling HTTP strict transport security

HTTP Strict Transport Security (HSTS) policy is a security enhancement, which ensures that only
HTTPS traffic is allowed on the host. Any HTTP requests are dropped by default. This is useful for
ensuring secure interactions with websites, or to offer a secure application for the user’s benefit.

When HSTS is enabled, HSTS adds a Strict Transport Security header to HTTPS responses from the
site. You can use the `insecureEdgeTerminationPolicy` value in a route to redirect to send HTTP to
HTTPS. However, when HSTS is enabled, the client changes all requests from the HTTP URL to HTTPS
before the request is sent, eliminating the need for a redirect. This is not required to be supported by the
client, and can be disabled by setting `max-age=0`.

**IMPORTANT**

HSTS works only with secure routes (either edge terminated or re-encrypt). The
configuration is ineffective on HTTP or passthrough routes.

Procedure

- To enable HSTS on a route, add the `haproxy.router.openshift.io/hsts_header` value to the
  edge terminated or re-encrypt route:
kind: Route
metadata:
  annotations:
    haproxy.router.openshift.io/hsts_header: max-age=31536000;includeSubDomains;preload

1. **max-age** is the only required parameter. It measures the length of time, in seconds, that the HSTS policy is in effect. The client updates **max-age** whenever a response with a HSTS header is received from the host. When **max-age** times out, the client discards the policy.

2. **includeSubDomains** is optional. When included, it tells the client that all subdomains of the host are to be treated the same as the host.

3. **preload** is optional. When **max-age** is greater than 0, then including **preload** in `haproxy.router.openshift.io/hsts_header` allows 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, before they have interacted with the site. Without **preload** set, browsers must have interacted with the site over HTTPS to get the header.

### 12.1.3. Troubleshooting throughput issues

Sometimes applications deployed through OpenShift Container Platform can cause network throughput issues such as unusually high latency between specific services.

Use the following methods to analyze performance issues if Pod logs do not reveal any cause of the problem:

- 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.

  ```bash
  $ tcpdump -s 0 -i any -w /tmp/dump.pcap host <podip 1> && host <podip 2> 1
  
  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.
  
  tcpdump generates a file at `/tmp/dump.pcap` containing all traffic between these two Pods. Ideally, 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:
  
  ```bash
  $ 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. Run the tool from the Pods first, then from the nodes, to locate any bottlenecks.
  
  For information on installing and using iperf, see this Red Hat Solution.

### 12.1.4. Using cookies to keep route statefulness

CHAPTER 12. CONFIGURING ROUTES

---

133
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.

12.1.4.1. 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 desired cookie name:

   $ oc annotate route <route_name> router.openshift.io/<cookie_name>="<cookie_annotation>"

   For example, to annotate the cookie name of my_cookie to the my_route with the annotation of my_cookie_annotation:

   $ oc annotate route my_route router.openshift.io/my_cookie="-my_cookie_annotation"

2. Save the cookie, and access the route:

   $ curl $my_route -k -c /tmp/my_cookie

12.1.5. 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.

Table 12.1. Route annotations

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Environment variable used as default</th>
</tr>
</thead>
<tbody>
<tr>
<td>haproxy.router.openshift.io/balance</td>
<td>Sets the load-balancing algorithm. Available options are source, roundrobin, and leastconn.</td>
<td>ROUTER_TCP_BALANCE_SCHEME for passthrough routes. Otherwise, use ROUTER_LOAD_BALANCE_ALGORITHM.</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
<td>Environment variable used as default</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td>-------------------------------------</td>
</tr>
<tr>
<td>haproxy.router.openshift.io/disable_cookies</td>
<td>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.</td>
<td></td>
</tr>
<tr>
<td>router.openshift.io/cookie_name</td>
<td>Specifies an optional cookie to use for this route. The name must consist of any combination of upper and lower case letters, digits, &quot;_&quot;, and &quot;-&quot;. The default is the hashed internal key name for the route.</td>
<td></td>
</tr>
<tr>
<td>haproxy.router.openshift.io/pod-concurrent-connections</td>
<td>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. But 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.</td>
<td></td>
</tr>
<tr>
<td>haproxy.router.openshift.io/rate-limit-connections</td>
<td>Setting true or TRUE to enables rate limiting functionality.</td>
<td></td>
</tr>
<tr>
<td>haproxy.router.openshift.io/rate-limit-connections.concurrent-tcp</td>
<td>Limits the number of concurrent TCP connections shared by an IP address.</td>
<td></td>
</tr>
<tr>
<td>haproxy.router.openshift.io/rate-limit-connections.rate-http</td>
<td>Limits the rate at which an IP address can make HTTP requests.</td>
<td></td>
</tr>
<tr>
<td>haproxy.router.openshift.io/rate-limit-connections.rate-tcp</td>
<td>Limits the rate at which an IP address can make TCP connections.</td>
<td></td>
</tr>
<tr>
<td>haproxy.router.openshift.io/timeout</td>
<td>Sets a server-side timeout for the route. (TimeUnits)</td>
<td>ROUTER_DEFAULT_SERVER_TIMEOUT</td>
</tr>
<tr>
<td>router.openshift.io/haproxy.health.check.interval</td>
<td>Sets the interval for the back-end health checks. (TimeUnits)</td>
<td>ROUTER_BACKEND_CHECK_INTERVAL</td>
</tr>
</tbody>
</table>
### Variable Description Environment variable used as default

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Environment variable used as default</th>
</tr>
</thead>
<tbody>
<tr>
<td>haproxy.router.openshift.io/ip_whitelist</td>
<td>Sets a whitelist for the route.</td>
<td></td>
</tr>
<tr>
<td>haproxy.router.openshift.io/hosts_header</td>
<td>Sets a Strict-Transport-Security header for the edge terminated or re-encrypt route.</td>
<td></td>
</tr>
</tbody>
</table>

**NOTE**

Environment variables can not be edited.

**A route setting custom timeout**

```yaml
apiVersion: v1
class: Route
data:
  annotations:
    haproxy.router.openshift.io/timeout: 5500ms

...  
```

1. 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.

### 12.1.6. 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 host name.

**WARNING**

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

**Prerequisites**
• Cluster administrator privileges.

**Procedure**

• Edit the `.spec.routeAdmission` field of the `ingresscontroller` resource variable using the following command:

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

**Sample Ingress Controller configuration**

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

### 12.2. SECURED ROUTES

The following sections describe how to create re-encrypt and edge 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.

### 12.2.1. 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` resource 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:

```bash
$ 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 host name for www.example.com.

- Create a secure Route resource using reencrypt TLS termination and a custom certificate:

```bash
$ 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**

```yaml
apiVersion: v1
category: 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.

12.2.2. 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` resource 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:

```bash
$ 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` resource that you want to expose for `frontend`. Substitute the appropriate host name for `www.example.com`.

- Create a secure `Route` resource using edge TLS termination and a custom certificate.

```bash
$ 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**

```yaml
apiVersion: 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: |
```
See `oc create route edge --help` for more options.
CHAPTER 13. CONFIGURING INGRESS CLUSTER TRAFFIC

13.1. 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.

<table>
<thead>
<tr>
<th>Method</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use an Ingress Controller</td>
<td>Allows access to HTTP/HTTPS traffic and TLS-encrypted protocols other than HTTPS (for example, TLS with the SNI header).</td>
</tr>
<tr>
<td>Automatically assign an external IP using a load balancer service</td>
<td>Allows traffic to non-standard ports through an IP address assigned from a pool.</td>
</tr>
<tr>
<td>Manually assign an external IP to a service</td>
<td>Allows traffic to non-standard ports through a specific IP address.</td>
</tr>
<tr>
<td>Configure a NodePort</td>
<td>Expose a service on all nodes in the cluster.</td>
</tr>
</tbody>
</table>

13.2. 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.

13.2.1. 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.

When a set of routes is created in various projects, the overall set of routes is available to the set of Ingress Controllers. Each Ingress Controller admits routes from the set of routes. By default, all Ingress Controllers admit all routes.

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.

### 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.

### 13.2.2. 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:

   ```
   $ oc new-project <project_name>
   ```

   For example:

   ```
   $ oc new-project myproject
   ```
2. Use the **oc new-app** command to create a service. For example:

```
$ oc new-app \
  -e MYSQL_USER=admin \n  -e MYSQL_PASSWORD=redhat \n  -e MYSQL_DATABASE=mysqldb \n  registry.redhat.io/rhscl/mysql-80-rhel7
```

3. Run the following command to see that the new service is created:

```
$ oc get svc -n myproject
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>TYPE</th>
<th>CLUSTER-IP</th>
<th>EXTERNAL-IP</th>
<th>PORT(S)</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>mysql-80-rhel7</td>
<td>ClusterIP</td>
<td>172.30.63.31</td>
<td>&lt;none&gt;</td>
<td>3306/TCP</td>
<td>4m55s</td>
</tr>
</tbody>
</table>

By default, the new service does not have an external IP address.

13.2.3. 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 project1
```

3. Run the following command to expose the route:

```
$ oc expose service <service_name>
```

For example:

```
$ oc expose service mysql-80-rhel7
route "mysql-80-rhel7" exposed
```

4. Use a tool, such as cURL, to make sure you can reach the service using the cluster IP address for the service:

```
$ curl <pod_ip>:<port>
```

For example:

```
$ curl 172.30.131.89:3306
```

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 connected to the service.

If you have a MySQL client, log in with the standard CLI command:
13.2.4. 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.

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:

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

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`.

13.2.5. 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.
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:

   ```yaml
   # cat router-internal.yaml
   apiVersion: v1
   items:
   - apiVersion: operator.openshift.io/v1
     kind: IngressController
     metadata:
       name: sharded
       namespace: openshift-ingress-operator
     spec:
       domain: <apps-sharded.basedomain.example.net>
       nodePlacement:
         nodeSelector:
           matchLabels:
             node-role.kubernetes.io/worker: ""  
           namespaceSelector:
             matchLabels:
               type: sharded
       status: {}
   kind: List
   metadata:
     resourceVersion: ""
     selfLink: ""
   
   2. Apply the Ingress Controller `router-internal.yaml` file:

   ```bash
   # 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**.

13.2.6. Additional resources

- The Ingress Operator manages wildcard DNS. For more information, see *Ingress Operator in OpenShift Container Platform*, *Installing a cluster on bare metal*, and *Installing a cluster on vSphere*.

**13.3. 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.

**13.3.1. 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.

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.

13.3.2. 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:

   $ oc new-project <project_name>

   For example:

   $ oc new-project myproject

2. Use the oc new-app command to create a service. For example:

   $ oc new-app
   -e MYSQL_USER=admin

-e MYSQL_PASSWORD=redhat \ 
-e MYSQL_DATABASE=mysqldb \ 
registry.redhat.io/rhscl/mysql-80-rhel7

3. Run the following command to see that the new service is created:

```bash
$ oc get svc -n myproject
NAME             TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)    AGE
mysql-80-rhel7   ClusterIP   172.30.63.31   <none>        3306/TCP   4m55s
```

By default, the new service does not have an external IP address.

### 13.3.3. 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:

```bash
$ oc project project1
```

3. Run the following command to expose the route:

```bash
$ oc expose service <service_name>
```

For example:

```bash
$ oc expose service mysql-80-rhel7
route "mysql-80-rhel7" exposed
```

4. Use a tool, such as cURL, to make sure you can reach the service using the cluster IP address for the service:

```bash
$ curl <pod_ip>:<port>
```

For example:

```bash
$ curl 172.30.131.89:3306
```

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 connected to the service.

If you have a MySQL client, log in with the standard CLI command:

```bash
$ mysql -h 172.30.131.89 -u admin -p
Enter password:
Welcome to the MariaDB monitor. Commands end with ; or \
g.
13.3.4. 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.

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 master node and paste the following text, editing the file as needed:

   **Sample load balancer configuration file**

   ```yaml
   apiVersion: v1
   kind: Service
   metadata:
     name: egress-2
   spec:
     ports:
       - name: db
         port: 3306
     loadBalancerIP:
       type: LoadBalancer
     selector:
       name: mysql
   ```

   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 `loadbalancer` as the type.
   4. Enter the name of the service.

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
   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
   Enter password:
   Welcome to the MariaDB monitor. Commands end with ; or \g.
   MySQL [(none)]>

### 13.4. CONFIGURING INGRESS CLUSTER TRAFFIC USING A SERVICE EXTERNAL IP

OpenShift Container Platform provides methods for communicating from outside the cluster with services running in the cluster. This method uses a service external IP.

#### 13.4.1. Using a service external IP to get traffic into the cluster

One method to expose a service is to assign an external IP address directly to the service you want to make accessible from outside the cluster.

The external IP address that you use must be provisioned on your infrastructure platform and attached to a cluster node.

With an external IP on the service, OpenShift Container Platform sets up NAT rules to allow traffic arriving at any cluster node attached to that IP address to be sent to one of the internal pods. This is similar to the internal service IP addresses, but the external IP tells OpenShift Container Platform that
this service should also be exposed externally at the given IP. The administrator must assign the IP address to a host (node) interface on one of the nodes in the cluster. Alternatively, the address can be used as a virtual IP (VIP).

These IPs are not managed by OpenShift Container Platform and administrators are responsible for ensuring that traffic arrives at a node with this IP.

![NOTE]

**NOTE**

The procedures in this section require prerequisites performed by the cluster administrator.

**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:

  ```bash
  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.

### 13.4.2. 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:

   ```bash
   $ oc new-project <project_name>
   ```

   For example:

   ```bash
   $ oc new-project myproject
   ```

2. Use the `oc new-app` command to create a service. For example:

   ```bash
   $ oc new-app \
   -e MYSQL_USER=admin \n   -e MYSQL_PASSWORD=redhat \n   ```
3. Run the following command to see that the new service is created:

```
$ oc get svc -n myproject
NAME       TYPE          CLUSTER-IP     EXTERNAL-IP   PORT(S)    AGE
mysql-80-rhel7  ClusterIP   172.30.63.31   <none>        3306/TCP   4m55s
```

By default, the new service does not have an external IP address.

### 13.4.3. 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 project1
   ```
3. Run the following command to expose the route:

   ```
   $ oc expose service <service_name>
   ```
   For example:

   ```
   $ oc expose service mysql-80-rhel7
   route "mysql-80-rhel7" exposed
   ```
4. Use a tool, such as cURL, to make sure you can reach the service using the cluster IP address for the service:

   ```
   $ curl <pod_ip>:<port>
   ```
   For example:

   ```
   $ curl 172.30.131.89:3306
   ```

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 connected to the service.

If you have a MySQL client, log in with the standard CLI command:

```
$ mysql -h 172.30.131.89 -u admin -p
Enter password:
Welcome to the MariaDB monitor. Commands end with ; or \g.
MySQL [(none)]>
```
13.5. 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.

13.5.1. 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 NodePorts requires additional port resources.

A NodePort exposes the service on a static port on the node’s IP address. NodePorts 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.

**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:

  ```bash
  $ 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.

13.5.2. 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:

   ```
   $ oc new-project <project_name>
   
   For example:
   
   $ oc new-project myproject
   ```

2. Use the **oc new-app** command to create a service. For example:

   ```
   $ oc new-app \ 
   -e MYSQL_USER=admin \ 
   -e MYSQL_PASSWORD=redhat \ 
   -e MYSQL_DATABASE=mysqldb \ 
   registry.redhat.io/rhscl/mysql-80-rhel7
   ```

3. Run the following command to see that the new service is created:

   ```
   $ oc get svc -n myproject
   
   NAME             TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)    AGE
   mysql-80-rhel7   ClusterIP   172.30.63.31   <none>        3306/TCP   4m55s
   ```

   By default, the new service does not have an external IP address.

**13.5.3. 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 project1
   ```

3. To expose a node port for the application, enter the following command. OpenShift Container Platform automatically selects an available port in the **30000-32767** range.

   ```
   $ oc expose dc mysql-80-rhel7 --type=NodePort --name=mysql-ingress
   ```

4. Optional: To confirm the service is available with a node port exposed, enter the following command:

   ```
   $ oc get svc -n myproject
   
   NAME             TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
   mysql-80-rhel7   ClusterIP   172.30.217.127   <none>        3306/TCP         9m44s
   mysql-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 mysql-80-rhel7
```
CHAPTER 14. 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.

**IMPORTANT**

The cluster-wide proxy is only supported if you used a user-provisioned infrastructure installation or provide your own networking, such as a virtual private cloud or virtual network, for a supported provider.

**Prerequisites**

- Review the sites that your cluster requires access to and determine whether any of them must bypass the proxy. By default, all cluster egress traffic is proxied, including calls to the cloud provider API for the cloud that hosts your cluster. Add sites to the Proxy object’s `spec.noProxy` field to bypass the proxy if necessary.

**NOTE**

The Proxy object’s `status.noProxy` field is populated by default with the instance metadata endpoint (169.254.169.254) and with the values of the `networking.machineNetwork[].cidr`, `networking.clusterNetwork[].cidr`, and `networking.serviceNetwork` fields from your installation configuration.

14.1. 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:

```yaml
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
Procedure

1. Create a ConfigMap 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:

   ```yaml
   apiVersion: v1
data:
   ca-bundle.crt: |
   <MY_PEM_ENCODED_CERTS>
kind: ConfigMap
metadata:
   name: user-ca-bundle
   namespace: openshift-config

   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 ConfigMap name that will be referenced from the Proxy object.
   4 The ConfigMap must be in the `openshift-config` namespace.

   b. Create the ConfigMap from this file:

   ```sh
   $ oc create -f user-ca-bundle.yaml
   ```

2. Use the `oc edit` command to modify the Proxy object:

   ```sh
   $ oc edit proxy/cluster
   ```

3. Configure the necessary fields for the proxy:

   ```yaml
   apiVersion: config.openshift.io/v1
   kind: Proxy
   metadata:
     name: cluster
   spec:
     httpProxy: http://<username>:<pswd>@<ip>:<port>
     httpsProxy: http://<username>:<pswd>@<ip>:<port>
     noProxy: example.com
     readinessEndpoints:
   ```
A proxy URL to use for creating HTTP connections outside the cluster. The URL scheme must be http.

A proxy URL to use for creating HTTPS connections outside the cluster. If this is not specified, then httpProxy is used for both HTTP and HTTPS connections. The URL scheme must be http; https is currently not supported.

A comma-separated list of destination domain names, domains, IP addresses or other network CIDRs to exclude proxying. Preface a domain with . to include all subdomains of that domain. Use * to bypass proxy for all destinations. Note that if you scale up workers not included in networking.machineNetwork[].cidr from the installation configuration, you must add them to this list to prevent connection issues.

One or more URLs external to the cluster to use to perform a readiness check before writing the httpProxy and httpsProxy values to status.

A reference to the ConfigMap in the openshift-config namespace that contains additional CA certificates required for proxying HTTPS connections. Note that the ConfigMap 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.

14.2. 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:

   ```yaml
   apiVersion: config.openshift.io/v1
   kind: Proxy
   metadata:
     name: cluster
   spec: {}
   status: {}
   ```
3. Save the file to apply the changes.
CHAPTER 15. 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.

15.1. 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**

- An existing `install-config.yaml` file.

- Review the sites that your cluster requires access to and determine whether any need to bypass the proxy. By default, all cluster egress traffic is proxied, including calls to hosting cloud provider APIs. Add sites to the Proxy object’s `spec.noProxy` field to bypass the proxy if necessary.

**NOTE**

The Proxy object’s `status.noProxy` field is populated by default with the instance metadata endpoint (169.254.169.254) and with the values of the `networking.machineNetwork[].cidr`, `networking.clusterNetwork[].cidr`, and `networking.serviceNetwork` fields from your installation configuration.
Procedure

1. Edit your `install-config.yaml` file and add the proxy settings. For example:

   ```yaml
   apiVersion: v1
   baseDomain: my.domain.com
   proxy:
     httpProxy: http://<username>:<pswd>@<ip>:<port>  
     httpsProxy: http://<username>:<pswd>@<ip>:<port> 
     noProxy: example.com
   additionalTrustBundle: |
     -----BEGIN CERTIFICATE-----
     <MY_TRUSTED_CA_CERT>
     -----END CERTIFICATE-----
   ...
   ```

   1. A proxy URL to use for creating HTTP connections outside the cluster. The URL scheme must be `http`. If you use an MITM transparent proxy network that does not require additional proxy configuration but requires additional CAs, you must not specify an `httpProxy` value.

   2. A proxy URL to use for creating HTTPS connections outside the cluster. If this field is not specified, then `httpProxy` is used for both HTTP and HTTPS connections. The URL scheme must be `http`, `https` is currently not supported. If you use an MITM transparent proxy network that does not require additional proxy configuration but requires additional CAs, you must not specify an `httpsProxy` value.

   3. A comma-separated list of destination domain names, domains, IP addresses, or other network CIDRs to exclude proxying. Preface a domain with . to include all subdomains of that domain. Use * to bypass proxy for all destinations.

   4. If provided, the installation program generates a ConfigMap 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` ConfigMap that merges these contents with the Red Hat Enterprise Linux CoreOS (RHCOS) trust bundle, and this ConfigMap is referenced in the Proxy object’s `trustedCA` field. The `additionalTrustBundle` field is required unless the proxy’s identity certificate is signed by an authority from the RHCOS trust bundle. If you use an MITM transparent proxy network that does not require additional proxy configuration but requires additional CAs, you must provide the MITM CA certificate.

   **NOTE**

   The installation program does not support the proxy `readinessEndpoints` field.

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.

15.2. 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:

```yaml
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 ConfigMap 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:

   ```yaml
   apiVersion: v1
data:
  ca-bundle.crt: | 1
  <MY_PEM_ENCODED_CERTS> 2
kind: ConfigMap
   ```
This data key must be named `ca-bundle.crt`.

One or more PEM-encoded X.509 certificates used to sign the proxy’s identity certificate.

The ConfigMap name that will be referenced from the Proxy object.

The ConfigMap must be in the `openshift-config` namespace.

b. Create the ConfigMap from this file:

```bash
$ oc create -f user-ca-bundle.yaml
```

2. Use the `oc edit` command to modify the Proxy object:

```bash
$ oc edit proxy/cluster
```

3. Configure the necessary fields for the proxy:

```yaml
metadata:
  name: user-ca-bundle
  namespace: openshift-config

apiVersion: config.openshift.io/v1
kind: Proxy
metadata:
  name: cluster
spec:
  httpProxy: http://<username>:<pswd>@<ip>:<port> # 1
  httpsProxy: http://<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. If this is not specified, then `httpProxy` is used for both HTTP and HTTPS connections. The URL scheme must be `http`. `https` is currently not supported.

3. A comma-separated list of destination domain names, domains, IP addresses or other network CIDs to exclude proxying. Preface a domain with `*` to include all subdomains of that domain. Use `*` to bypass proxy for all destinations. Note that if you scale up workers not included in `networking.machineNetwork[].cidr` from the installation configuration, you must add them to this list to prevent connection issues.

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.
A reference to the ConfigMap in the `openshift-config` namespace that contains additional CA certificates required for proxying HTTPS connections. Note that the ConfigMap must

4. Save the file to apply the changes.

## 15.3. 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.

Operators request this injection by creating an empty ConfigMap with the following label:

```bash
config.openshift.io/inject-trusted-cabundle="true"
```

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:

```yaml
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
                  path: tls-ca-bundle.pem

1 ca-bundle.crt is required as the ConfigMap key.
2. `tls-ca-bundle.pem` is required as the ConfigMap path.