OpenShift Container Platform 4.2

Networking

Configuring and managing cluster 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 ................................. 6
  1.1. OPENSHIFT CONTAINER PLATFORM DNS 6

## CHAPTER 2. ACCESSING HOSTS ............................................. 7
  2.1. ACCESSING HOSTS ON AMAZON WEB SERVICES (AWS) AFTER A FULL-STACK AUTOMATION (IPI) INSTALLATION 7

## CHAPTER 3. CLUSTER NETWORK OPERATOR IN OPENSHIFT CONTAINER PLATFORM .......... 8
  3.1. CLUSTER NETWORK OPERATOR 8
  3.2. VIEWING THE CLUSTER NETWORK CONFIGURATION 8
  3.3. VIEWING CLUSTER NETWORK OPERATOR STATUS 9
  3.4. VIEWING CLUSTER NETWORK OPERATOR LOGS 9
  3.5. CLUSTER NETWORK OPERATOR CUSTOM RESOURCE (CR)
      3.5.1. Configuration parameters for OpenShift SDN 10
      3.5.2. Cluster Network Operator example CR 11

## CHAPTER 4. DNS OPERATOR IN OPENSHIFT CONTAINER PLATFORM .................. 12
  4.1. DNS OPERATOR 12
  4.2. VIEW THE DEFAULT DNS 12
  4.3. DNS OPERATOR STATUS 13
  4.4. DNS OPERATOR LOGS 13

## CHAPTER 5. INGRESS OPERATOR IN OPENSHIFT CONTAINER PLATFORM ............... 14
  5.1. THE INGRESS CONFIGURATION ASSET 14
  5.2. INGRESS CONTROLLER CONFIGURATION PARAMETERS 14
      5.2.1. Ingress controller endpoint publishing strategy 16
  5.3. VIEW THE DEFAULT INGRESS CONTROLLER 16
  5.4. VIEW INGRESS OPERATOR STATUS 17
  5.5. VIEW INGRESS CONTROLLER LOGS 17
  5.6. VIEW INGRESS CONTROLLER STATUS 17
  5.7. SETTING A CUSTOM DEFAULT CERTIFICATE 17
  5.8. SCALING AN INGRESS CONTROLLER 18
  5.9. CONFIGURING INGRESS CONTROLLER SHARDING BY USING ROUTE LABELS 19
  5.10. CONFIGURING INGRESS CONTROLLER SHARDING BY USING NAMESPACE LABELS 20
  5.11. ADDITIONAL RESOURCES 21

## CHAPTER 6. CONFIGURING NETWORK POLICY WITH OPENSHIFT SDN .................. 22
  6.1. ABOUT NETWORK POLICY 22
  6.2. EXAMPLE NETWORKPOLICY OBJECT 23
  6.3. CREATING A NETWORKPOLICY OBJECT 24
  6.4. DELETING A NETWORKPOLICY OBJECT 25
  6.5. VIEWING NETWORKPOLICY OBJECTS 25
  6.6. CONFIGURING MULTITENANT ISOLATION USING NETWORKPOLICY 26
  6.7. CREATING DEFAULT NETWORK POLICIES FOR A NEW PROJECT 27
      6.7.1. Modifying the template for new projects 27
      6.7.2. Adding network policy objects to the new project template 28

## CHAPTER 7. MULTIPLE NETWORKS ........................................... 30
  7.1. UNDERSTANDING MULTIPLE NETWORKS 30
      7.1.1. Usage scenarios for an additional network 30
      7.1.2. Additional networks in OpenShift Container Platform 30
  7.2. ATTACHING A POD TO AN ADDITIONAL NETWORK 31
      7.2.1. Adding a Pod to an additional network 31
7.3. REMOVING A POD FROM AN ADDITIONAL NETWORK

7.3.1. Removing a Pod from an additional network

7.4. CONFIGURING A BRIDGE NETWORK

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

7.4.1.1. Configuration for bridge

7.4.1.1.1. bridge configuration example

7.4.1.2. Configuration for ipam CNI plug-in

7.4.1.2.1. Static IP address assignment configuration example

7.4.1.2.2. Dynamic IP address assignment configuration example

7.5. CONFIGURING A MACVLAN NETWORK

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

7.5.1.1. Configuration for macvlan CNI plug-in

7.5.1.1.1. macvlan configuration example

7.5.1.2. Configuration for ipam CNI plug-in

7.5.1.2.1. Static ipam configuration YAML

7.5.1.2.2. Dynamic ipam configuration YAML

7.5.1.2.3. Static IP address assignment configuration example

7.5.1.2.4. Dynamic IP address assignment configuration example

7.6. CONFIGURING AN IPVLAN NETWORK

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

7.6.1.1. Configuration for ipvlan

7.6.1.1.1. ipvlan configuration example

7.6.1.2. Configuration for ipam CNI plug-in

7.6.1.2.1. Static IP address assignment configuration example

7.6.1.2.2. Dynamic IP address assignment configuration example

7.7. CONFIGURING A HOST-DEVICE NETWORK

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

7.7.1.1. Configuration for host-device

7.7.1.1.1. host-device configuration example

7.7.1.2. Configuration for ipam CNI plug-in

7.7.1.2.1. Static IP address assignment configuration example

7.7.1.2.2. Dynamic IP address assignment configuration example

7.8. CONFIGURING AN ADDITIONAL NETWORK FOR SR-IOV

7.8.1. About SR-IOV hardware on OpenShift Container Platform

7.8.1.1. Supported devices

7.8.1.2. Automated discovery of SR-IOV network devices

7.8.1.3. Example use of virtual function (VF) in a Pod

7.8.2. Installing SR-IOV Network Operator

7.8.2.1. Installing the Operator using the CLI

7.8.2.2. Installing the Operator using the web console

7.8.3. Configuring SR-IOV network devices

7.8.4. Configuring SR-IOV additional network

7.8.4.1. Configuration for ipam CNI plug-in

7.8.4.1.1. Static IP address assignment configuration example

7.8.4.1.2. Dynamic IP address assignment configuration example

7.8.5. Adding a Pod to an additional network

7.9. EDITING AN ADDITIONAL NETWORK

7.9.1. Modifying an additional network attachment definition

7.10. REMOVING AN ADDITIONAL NETWORK

7.10.1. Removing an additional network attachment definition

CHAPTER 8. OPENSSHIFT SDN

8.1. ABOUT OPENSSHIFT SDN
8.2. ASSIGNING EGRESS IPS TO A PROJECT
  8.2.1. Enabling automatically assigned egress IPs for a namespace
  8.2.2. Configuring manually assigned egress IPs
8.3. USING MULTICAST
  8.3.1. About multicast
  8.3.2. Enabling multicast between Pods
  8.3.3. Disabling multicast between Pods
8.4. CONFIGURING NETWORK ISOLATION USING OPENSHELP SDN
  8.4.1. Joining projects
  8.4.2. Isolating a project
  8.4.3. Disabling network isolation for a project
8.5. CONFIGURING KUBE-PROXY
  8.5.1. About iptables rules synchronization
  8.5.2. Modifying the kube-proxy configuration
  8.5.3. kube-proxy configuration parameters

CHAPTER 9. CONFIGURING ROUTES ................................................................. 78
  9.1. ROUTE CONFIGURATION
    9.1.1. Configuring route timeouts
    9.1.2. Enabling HTTP strict transport security
    9.1.3. Troubleshooting throughput issues
    9.1.4. Using cookies to keep route statefulness
    9.1.4.1. Annotating a route with a cookie
  9.2. SECURED ROUTES
    9.2.1. Creating a re-encrypt route with a custom certificate
    9.2.2. Creating an edge route with a custom certificate

CHAPTER 10. CONFIGURING INGRESS CLUSTER TRAFFIC ................................. 84
  10.1. CONFIGURING INGRESS CLUSTER TRAFFIC OVERVIEW ............................ 84
  10.2. CONFIGURING INGRESS CLUSTER TRAFFIC USING AN INGRESS CONTROLLER
    10.2.1. Using Ingress Controllers and routes
    10.2.2. Creating a project and service
    10.2.3. Exposing the service by creating a route
    10.2.4. Configuring ingress controller sharding by using route labels
    10.2.5. Configuring ingress controller sharding by using namespace labels
    10.2.6. Additional resources
  10.3. CONFIGURING INGRESS CLUSTER TRAFFIC USING A LOAD BALANCER
    10.3.1. Using a load balancer to get traffic into the cluster
    10.3.2. Creating a project and service
    10.3.3. Exposing the service by creating a route
    10.3.4. Creating a load balancer service
  10.4. CONFIGURING INGRESS CLUSTER TRAFFIC USING A SERVICE EXTERNAL IP
    10.4.1. Using a service external IP to get traffic into the cluster
    10.4.2. Creating a project and service
    10.4.3. Exposing the service by creating a route
  10.5. CONFIGURING INGRESS CLUSTER TRAFFIC USING A NODEPORT
    10.5.1. Using a NodePort to get traffic into the cluster
    10.5.2. Creating a project and service
    10.5.3. Exposing the service by creating a route

CHAPTER 11. CONFIGURING THE CLUSTER-WIDE PROXY ................................... 98
  11.1. ENABLING THE CLUSTER-WIDE PROXY
  11.2. REMOVING THE CLUSTER-WIDE PROXY
CHAPTER 12. CONFIGURING A CUSTOM PKI .......................................................... 102
12.1. CONFIGURING THE CLUSTER-WIDE PROXY DURING INSTALLATION 102
12.2. ENABLING THE CLUSTER-WIDE PROXY ............................................... 103
12.3. CERTIFICATE INJECTION USING OPERATORS ....................................... 105
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.

1.1. OPENSShift 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 (AWS) AFTER A FULL-STACK AUTOMATION (IPI) INSTALLATION

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 OPENSSHIFT CONTAINER PLATFORM

The Cluster Network Operator (CNO) deploys and manages the cluster network components on an OpenShift Container Platform cluster, including the Container Network Interface (CNI) Software Defined Networking (SDN) 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 plug-in, or a different SDN plug-in if selected during cluster installation, 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 4.2.0 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: 
   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
  ```

### 3.5. CLUSTER NETWORK OPERATOR CUSTOM RESOURCE (CR)

The cluster network configuration in the `Network.operator.openshift.io` custom resource (CR) stores the configuration settings for the Cluster Network Operator (CNO). The Operator manages the cluster network.

You can specify the cluster network configuration for your OpenShift Container Platform cluster by setting the parameters 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 valid parameter values:
Cluster Network Operator CR

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: 5s
    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 software-defined networking (SDN) for the cluster network.

4. The parameters for this object specify the Kubernetes network proxy (kube-proxy) configuration.

5. 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.

6. 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 OpenShift SDN

The following YAML object describes the configuration parameters for OpenShift SDN:

defaultNetwork:
  type: OpenShiftSDN 1
  openshiftSDNConfig: 2
    mode: NetworkPolicy 3
  mtu: 1450 4
  vxlanPort: 4789 5

1. The Software Defined Networking (SDN) plug-in being used. OpenShift SDN is the only plug-in supported in OpenShift Container Platform 4.2.

2. OpenShift SDN specific configuration parameters.
The network isolation mode for the OpenShift SDN CNI plug-in.

MTU for the VXLAN overlay network. This value is normally configured automatically.

The port to use for all VXLAN packets. The default value is 4789.

3.5.2. Cluster Network Operator example CR

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

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

   ```
   $ 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:

   ```
   $ 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`. It cannot be customized, replaced, or supplemented with additional `dnses`.

Procedure

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

   ```
   $ oc describe dns.operator/default
   Name:         default
   Namespace:    default
   Labels:       <none>
   Annotations:  <none>
   API Version:  operator.openshift.io/v1
   Kind:         DNS
   ...
   Status:
   ```
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:

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

**NOTE**

Configuration of the CoreDNS Corefile or Kubernetes plugin is not supported.

### 4.3. 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
```

### 4.4. 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 OPENShift 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: config.openshift.io/v1</td>
<td></td>
</tr>
<tr>
<td>kind: Ingress</td>
<td></td>
</tr>
<tr>
<td>metadata:</td>
<td></td>
</tr>
<tr>
<td>name: cluster</td>
<td></td>
</tr>
<tr>
<td>spec:</td>
<td></td>
</tr>
<tr>
<td>domain: apps.openshiftdemos.com</td>
<td></td>
</tr>
</tbody>
</table>
The **domain** value must be unique among all Ingress controllers and cannot be updated.

If empty, the default value is `ingress.config.openshift.io/cluster.spec.domain`.

### replicas

The **replicas** value is the desired number of Ingress controller replicas. If not set, the default value is 2.

### endpointPublishingStrategy

The **endpointPublishingStrategy** value is used to publish the Ingress controller endpoints to other networks, enable load balancer integrations, and provide access to other systems.

If not set, the default value is based on `infrastructure.config.openshift.io/cluster.status.platform`:

- AWS: **LoadBalancerService** (with external scope)
- Azure: **LoadBalancerService** (with external scope)
- GCP: **LoadBalancerService** (with external scope)
- Other: **HostNetwork**.

The **endpointPublishingStrategy** value cannot be updated.

### defaultCertificate

The **defaultCertificate** value is a reference to a secret that contains the default certificate that is served by the Ingress controller. When Routes do not specify their own certificate, **defaultCertificate** is used.

The secret must contain the following keys and data:
- `tls.crt`: certificate file contents
- `tls.key`: key file contents

If not set, a wildcard certificate is automatically generated and used. The certificate is valid for the Ingress controller **domain** and **subdomains**, and the generated certificate’s CA is automatically integrated with the cluster’s trust store.

The in-use certificate, whether generated or user-specified, is automatically integrated with OpenShift Container Platform built-in OAuth server.
### Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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:

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

All parameters are optional.

### 5.2.1. Ingress controller endpoint publishing strategy

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 have configured.
- Your certificate is valid for the Ingress domain.
- You must have an IngressController CR. You may use the default one:
WARNING
If the default certificate is replaced, it must be signed by a public certificate authority already included in the CA bundle as provided by the container userspace.

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 \"{"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:

   ```bash
   $ 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:

   ```bash
   $ 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:

   ```bash
   $ 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:

   ```yaml
   # cat router-internal.yaml
   apiVersion: v1
   items:
   - apiVersion: operator.openshift.io/v1
     kind: IngressController
     metadata:
       name: sharded
       namespace: openshift-ingress-operator
     spec:
   ```
2. Apply the Ingress Controller `router-internal.yaml` file:

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

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:

```
# 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. ADDITIONAL RESOURCES

- Configuring a custom PKI
CHAPTER 6. CONFIGURING NETWORK POLICY WITH OPENSHIFT SDN

6.1. ABOUT NETWORK POLICY

In a cluster using a Kubernetes Container Network Interface (CNI) plug-in that supports NetworkPolicy, network isolation is controlled entirely by NetworkPolicy objects. In OpenShift Container Platform 4.2, 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.

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
  kind: NetworkPolicy
  metadata:
    name: allow-from-openshift-ingress
  spec:
    ingress:
    - from:
      namespaceSelector:
        matchLabels:
          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:

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

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

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-http-and-https
spec:
  podSelector:
    matchLabels:
      role: frontend
  ingress:
    - ports:
      - protocol: TCP
        port: 80
    - ports:
      - protocol: TCP
        port: 443
```

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:

```
kind: NetworkPolicy
apiVersion: extensions/v1beta1
metadata:
  name: allow-27107
```
### 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
   spec:
     podSelector:  
       matchLabels:  
         app: mongodb
     ingress:
       - from:
         - podSelector:  
           matchLabels:  
             app: app
           ports:  
             - protocol: TCP
             port: 27017
   
   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:

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

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

```bash
$ 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>
  ```

  1. 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:

```
$ 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. 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:

   ```yaml
   $ oc get networkpolicy allow-from-openshift-ingress -o yaml
   apiVersion: extensions/v1beta1
   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:
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:
$ 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-namespaces
     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
   
   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-namespaces     <none>         7s
   ```
CHAPTER 7. MULTIPLE NETWORKS

7.1. 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.

7.1.1. 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.

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

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

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

#### 7.2.1. 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 to an additional network, 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.

   ```shell
   $ 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 comma separated string of one or more NetworkAttachmentDefinition Custom Resource (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 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
class: Pod
metadata:
  name: example-pod
  annotations:
    k8s.v1.cni.cncf.io/networks: net1,net2
```
3. Optional: Confirm that the annotation exists in the Pod CR by running the following command. Replace `<name>` with the name of the Pod.

```bash
$ oc get pod <name> -o yaml
```

In the following example, the `example-pod` Pod is attached to the `net1` additional network:

```yaml
apiVersion: v1
kind: Pod
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: macvlan-bridge
    k8s.v1.cni.cncf.io/networks-status: |
      |
        |
          |
            |
              |
                |
                  |
                    |
                      |
                        |
                          |
                            |
                                |
                                    |
                                        |
                                            |
                                                |
                                                    |
                                                        |
                                                            |
                                                                |
                                                                    |
                                                                            |
                                                                                                                                                                                                                                                        
                                                                                                               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.

### 7.3. REMOVING A POD FROM AN ADDITIONAL NETWORK

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

#### 7.3.1. 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.

   ```bash
   $ 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:

     ```yaml
     apiVersion: v1
     kind: Pod
     metadata:
       name: example-pod
     annotations: {}
     spec:
       containers:
         - name: example-pod
           command: "/bin/bash", "-c", "sleep 2000000000000"
           image: centos/tools
     ```

   - To remove a specific additional network from a Pod, update the `k8s.v1.cni.cncf.io/networks` parameter by removing the name of the NetworkAttachmentDefinition for the additional network.

3. Optional: Confirm that the Pod is no longer attached to the additional network by running the following command. Replace `<name>` with the name of the Pod.

   ```bash
   $ oc describe pod <name>
   ``

   In the following example, the `example-pod` Pod is attached only to the default cluster network.

   ```bash
   $ oc describe pod example-pod
   Name:               example-pod
   ...  
   Annotations:   k8s.v1.cni.cncf.io/networks-status:
     []
   ```
7.4. 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.

7.4.1. 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:

   $ 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:
   ```
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
   NAME       AGE
   example-network  14m
   example-macvlan   21m
   ```

### 7.4.1.1. 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
additionalNetworks:
  - name: test-network-1
    type: Raw
    rawCNIConfig:
      
      "cniVersion": "0.3.1",
      "type": "bridge",
      "master": "eth1",
      "ipam": {
        "type": "static",
        "addresses": [
          
          "address": "192.168.1.1/24"
        ]
      }
```

1. Specify the configuration for the additional network attachment definition.
2. Name: `<namespace>`
3. Name: `<name>`
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": {
        ... 3
        "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.

2. 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**.

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

4. 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**.

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

6. 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.

7. 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**.

8. 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.

7.4.1.1. bridge configuration example

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

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

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

7.4.1.2. Configuration for ipam CNI plug-in

The IP address management (IPAM) 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.

**IMPORTANT**

In OpenShift Container Platform 4.2.0, if you attach a Pod to an additional network that uses DHCP for IP address management, the Pod will fail to start. This is fixed in OpenShift Container Platform 4.2.1. For more information, see BZ#1754686.

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": [ 
      
    ]
  }
}
```

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

2. The addresses parameter is required for dynamic IP address assignment.
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 to use to route network traffic to.

The DNS configuration. Optional.

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 will be 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.

7.4.1.2.1. Static IP address assignment configuration example

You can configure ipam for static IP address assignment:

```json
{
   "ipam": {
      "type": "static",
      "addresses": [
```
7.4.1.2.2. 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.

### 7.5. 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 policies of your cloud provider.

#### 7.5.1. 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:

   $ 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:
     - name: example-addn-network
       type: SimpleMacvlan
       simpleMacvlanConfig:
         ipamConfig:
           type: static
           staticPAMConfig:
             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.

   $ oc get network-attachment-definitions
   NAME       AGE
   example-network  14m
   example-macvlan  21m

7.5.1.1. 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

   name: <name>  
   namespace: <namespace>  
   type: SimpleMacvlan  
   simpleMacvlanConfig:
     master: <master>  
     mode: <mode>  
     mtu: <mtu>  
     ipamConfig:  
...
Specify a name for the additional network attachment that you are creating. The name must be unique within the specified namespace.

Specify the namespace to create the network attachment in. If a value is not specified, the default namespace is used.

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.

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.

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.

### 7.5.1.1.1. macvlan configuration example

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

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

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

The IP address management (IPAM) 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.

**IMPORTANT**

In OpenShift Container Platform 4.2.0, if you attach a Pod to an additional network that uses DHCP for IP address management, the Pod will fail to start. This is fixed in OpenShift Container Platform 4.2.1. For more information, see BZ#1754686.

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

### 7.5.1.2.1. 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.
6. The gateway to use to route network traffic to.
7. The DNS configuration. Optional.
8. A collection of one or more IP addresses for to send DNS queries to.
9. 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` will be rewritten as `example-host.example.com`.
10. An array of domain names to append to an unqualified host name, such as `example-host`, during a DNS lookup query.

### 7.5.1.2.2. Dynamic ipam configuration YAML
The following YAML describes a configuration for static IP address assignment:

**Dynamic ipam configuration YAML**

```yaml
ipamConfig:
  type: DHCP
```

### 7.5.1.2.3. Static IP address assignment configuration example

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

```yaml
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
```

### 7.5.1.2.4. Dynamic IP address assignment configuration example

The following example shows an ipam configuration for DHCP:

```yaml
ipamConfig:
  type: DHCP
```

**Next steps**

- Attach a Pod to an additional network.

## 7.6. 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.

### 7.6.1. 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:

   ```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 ipvlan CNI plug-in:

   ```yaml
   apiVersion: operator.openshift.io/v1
   kind: Network
   metadata:
     name: cluster
   spec:
     additionalNetworks:
     - name: test-network-1
       type: Raw
       rawCNIConfig: '{
         "cniVersion": "0.3.1",
         "type": "ipvlan",
         "master": "eth1",
         "mode": "l2",
         "ipam": {
           "type": "static",
           "addresses": [
             {
               "address": "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
   NAME                  AGE
   ```
7.6.1.1. 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>  # Specify a name for the additional network attachment that you are creating. The name must be unique within the specified namespace.
namespace: <namespace>  # Specify the namespace to create the network attachment in. If you do not specify a value, then the default namespace is used.
rawCNIConfig: '{  # Specify the CNI plug-in configuration in JSON format, which is based on the following template.
    ... 
}  
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
{  
    "cniVersion": "0.3.1",  
    "name": "<name>",  
    "type": "ipvlan",  
    "mode": "<mode>",  
    "master": "<master>",  
    "mtu": <mtu>,  
    "ipam": {  
        ...  
    }  
}
```

1. 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.

7.6.1.1.1. ipvlan configuration example

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

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

The CNI configuration object is specified as a YAML string.

7.6.1.2. Configuration for ipam CNI plug-in

The IP address management (IPAM) 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.

**IMPORTANT**

In OpenShift Container Platform 4.2.0, if you attach a Pod to an additional network that uses DHCP for IP address management, the Pod will fail to start. This is fixed in OpenShift Container Platform 4.2.1. For more information, see BZ#1754686.

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.

```json
ipam CNI plug-in JSON configuration object
{
}
```
"ipam": {
  "type": "<type>"  \(^1\)
  "addresses": [
    {
      "address": "<address>"  \(^2\)
      "gateway": "<gateway>"  \(^3\)
    }
  ],
  "routes": [
    {
      "dst": "<dst>"  \(^4\)
      "gw": "<gw>"  \(^5\)
    }
  ],
  "dns": {
    "nameservers": ["<nameserver>"],
    "domain": "<domain>"  \(^6\)
    "search": ["<search_domain>"]  \(^7\)
  }
}

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 to use to route network traffic to.

8. The DNS configuration. Optional.

9. An of 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** will be rewritten as **example-host.example.com**.

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

### 7.6.1.2.1. Static IP address assignment configuration example

You can configure ipam for static IP address assignment:
7.6.1.2.2. Dynamic IP address assignment configuration example

You can configure ipam for DHCP:

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

Next steps

- Attach a Pod to an additional network.

7.7. 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.

7.7.1. 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
    type: Raw
    rawCNIConfig: '{
      "cniVersion": "0.3.1",
      "type": "host-device",
      "device": "eth1"
    }'
```

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.

   ```bash
   $ oc get network-attachment-definitions
   NAME                 AGE
   example-network      14m
   example-macvlan      21m
   ```

7.7.1.1 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> 1
namespace: <namespace> 2
rawCNIConfig: '{ 3
```
Specify a name for the additional network attachment that you are creating. The name must be unique within the specified namespace.

Specify the namespace to create the network attachment in. If you do not specify a value, then the default namespace is used.

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.

### 7.7.1.1. host-device configuration example

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

```yaml
name: hostdev-net
```
The CNI configuration object is specified as a YAML string.

7.7.1.2. Configuration for ipam CNI plug-in

The IP address management (IPAM) 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.

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

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.

**IMPORTANT**

In OpenShift Container Platform 4.2.0, if you attach a Pod to an additional network that uses DHCP for IP address management, the Pod will fail to start. This is fixed in OpenShift Container Platform 4.2.1. For more information, see **BZ#1754686**.
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 to use to route network traffic to.

8. The DNS configuration. Optional.

9. An of 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** will be rewritten as **example-host.example.com**.

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

### 7.7.1.2.1. Static IP address assignment configuration example

You can configure ipam for static IP address assignment:

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

### 7.7.1.2.2. Dynamic IP address assignment configuration example

You can configure ipam for DHCP:

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

### Next steps
7.8. CONFIGURING AN ADDITIONAL NETWORK FOR SR-IOV

IMPORTANT

Network Interface Card (NIC) SR-IOV 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/.

7.8.1. 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 provides following features:

- Discover the SR-IOV network device in cluster.
- Initialize the supported SR-IOV NIC models on nodes.
- Provision the SR-IOV network device plug-in on nodes.
- Provision the SR-IOV CNI plug-in executable on nodes.
- Provision the Network Resources Injector in cluster.
- Manage configuration of SR-IOV network device plug-in.
- Generate NetworkAttachmentDefinition custom resources (CR) for the SR-IOV CNI plug-in.

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

- 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.
- 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 Network Resources Injector is enabled by default and cannot be disabled.

### 7.8.1.1. Supported devices

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

- Intel XXV710-DA2 25G card with vendor ID 0x8086 and device ID 0x158b
- Mellanox MT27710 Family [ConnectX-4 Lx] 25G card with vendor ID 0x15b3 and device ID 0x1015
- Mellanox MT27800 Family [ConnectX-5] 100G card with vendor ID 0x15b3 and device ID 0x1017

### 7.8.1.2. 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 `.spec.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:

```yaml
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodeState
metadata:
  name: node-25
  namespace: sriov-network-operator
  ownerReferences:
    - apiVersion: sriovnetwork.openshift.io/v1
      blockOwnerDeletion: true
      controller: true
      kind: SriovNetworkNodePolicy
      name: default
spec:
dpConfigVersion: d41d8cd98f00b204e9800998ecf8427e
status:
  interfaces:
    - deviceId: "1017"
      driver: mlx5_core
      mtu: 1500
      name: ens785f0
      pciAddress: "0000:18:00.0"
      totalvfs: 8
      vendor: 15b3
    - deviceId: "1017"
```
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.

7.8.1.3. Example use of 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. In the following example, a Pod is using a VF in RDMA mode:

```
apiVersion: v1
group: kind: Pod

metadata:
  name: rdma-app

annotations:
  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 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:
  - name: testpmd
    image: <DPDK_image>
    securityContext:
      capabilities:
        add: ["IPC_LOCK"]
    volumeMounts:
      - mountPath: /dev/hugepages
        name: hugepage
    resources:
      limits:
        memory: "1Gi"
        cpu: "2"
        hugepages-1Gi: "4Gi"
    requests:
      memory: "1Gi"
      cpu: "2"
      hugepages-1Gi: "4Gi"
    command: ["sleep", "infinity"]
  volumes:
  - name: hugepage
    emptyDir:
      medium: HugePages
```

### 7.8.2. 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.

#### 7.8.2.1. 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 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 `sriov-network-operator` namespace, and then save the YAML in the `sriov-namespace.yaml` file:

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

b. Create the namespace by running the following command:

```
$ 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: sriov-network-operator
spec:
  targetNamespaces:
  - sriov-network-operator
```

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

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

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

```
$ oc get packagemanifest sriov-network-operator -n openshift-marketplace -o jsonpath='{.status.channels[?(@.name=="alpha")].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: sriov-network-operator
spec:
  channel: <channel> ①
  name: sriov-network-operator
  source: redhat-operators ②
  sourceNamespace: openshift-marketplace
```
Specify the value from you obtained in the previous step for the `.status.channels[].name` parameter.

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 `sriov-network-operator` project:

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

Now using project "sriov-network-operator"

### 7.8.2.2. 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 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**.
   
   c. On the **Create Operator Subscription** page, under *A specific namespace on the cluster* select `sriov-network-operator`. Then, click **Subscribe**.

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 `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 sriov-network-operator project.

### 7.8.3. 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 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 sriov-network-operator namespace will change to a Running status.

### 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

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> 1
  namespace: sriov-network-operator 2
spec:
  resourceName: <sriov_resource_name> 3
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true" 4
  priority: <priority> 5
  mtu: <mtu> 6
  numVFs: <num> 7
  nicSelector: 8
  vendor: "<vendor_code>" 9
  deviceID: "<device_id>" 10
  pfName: ["<pf_name>", ...] 11
```
Specify a name for the CR.

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

Specify the resource name of the SR-IOV device plug-in. The prefix `openshift.io/` will be added when it’s referred in Pod spec. You can create multiple `SriovNetworkNodePolicy` CRs for a resource name.

Specify the node selector to select which node to be configured. User can choose to label the nodes manually or with tools like Kubernetes Node Feature Discovery. Only SR-IOV network devices on selected nodes will be configured. The SR-IOV CNI plug-in and device plug-in will be only deployed on selected nodes.

Specify 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.

Specify a value for the maximum transmission unit (MTU) of the virtual function. The value for MTU must be in the range from 1 to 9000. If you do not need to specify the MTU, specify a value of "."

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 `pfName`. If you specify both `pfName` and `rootDevices` at the same time, ensure that they point to an identical device.

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

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

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`.

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`.

Specify whether to enable RDMA mode. The default value is `false`. Only RDMA over Converged Ethernet (RoCE) mode is supported on Mellanox Ethernet adapters.
If RDMA 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 CR by running the following command:

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

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

### 7.8.4. 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.

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

   1. Replace `<name>` with a name for the CR. The Operator will create a NetworkAttachmentDefinition CR with same name.
   2. Specify the namespace where the SR-IOV Operator is installed.
   3. Replace `<target_namespace>` with the namespace where the NetworkAttachmentDefinition CR will be created.
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.

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 SriovNetworkNodePolicy CR that defines the SR-IOV hardware for this additional network.

2. Create the CR by running the following command:

```bash
$ 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.

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

7.8.4.1. Configuration for ipam CNI plug-in

The IP address management (IPAM) CNI plug-in manages IP address assignment for other CNI plugins. 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.

**IMPORTANT**

In OpenShift Container Platform 4.2.0, if you attach a Pod to an additional network that uses DHCP for IP address management, the Pod will fail to start. This is fixed in OpenShift Container Platform 4.2.1. For more information, see BZ#1754686.

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>",
            }
        ]
    }
}
```
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 to use to route network traffic to.

The DNS configuration. Optional.

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` will be 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.

### 7.8.4.1.1. Static IP address assignment configuration example

You can configure ipam for static IP address assignment:

```yaml

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

```
7.8.4.1.2. Dynamic IP address assignment configuration example

You can configure ipam for DHCP:

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

7.8.5. 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 to an additional network, 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.

   ```
   $ 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 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

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
```

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:

```yaml
$ 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: |
    |
    |
    [{
      "name": "openshift-sdn",
      "interface": "eth0",
      "ips": ["10.128.2.14"
        ],
      "default": true,
      "dns": {} }
    ],
    [{
      "name": "macvlan-bridge",
      "interface": "net1",
      "ips": ["20.2.2.100"
        ],
      "mac": "22:2f:60:a5:f8:00",
      "dns": {} }
    ]
  name: example-pod
  namespace: default
spec:
  ...
status:
  ...
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.

7.9. EDITING AN ADDITIONAL NETWORK

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

7.9.1. 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\n" .spec.config}}'
   {
     "cniVersion": "0.3.1",
     "type": "macvlan",
     "master": "ens5",
     "mode": "bridge",
     "ipam": {
       "type": "static",
       "routes": [{"dst": "0.0.0.0/0","gw": "10.128.2.1"}]
     },
     "addresses": [{
       "address": "10.128.2.100/23",
       "gateway": "10.128.2.1"
     }],
     "dns": {
       "nameservers": [
         "172.30.0.10"
       ],
       "domain": "us-west-2.compute.internal",
       "search": ["us-west-2.compute.internal"]
     }
   }
   ```
7.10. REMOVING AN ADDITIONAL NETWORK

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

7.10.1. 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.

IMPORTANT

In OpenShift Container Platform 4.2.0 you must manually delete the additional network CR after removing it from the Cluster Network Operator configuration. This will be fixed in a future release. For more information, see BZ#1755908.

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.

   ```json
   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. Delete the NetworkAttachmentDefinition CR for the additional network by running the following command. Replace `<name>` with the name of the additional network to remove.

   ```
   $ oc delete network-attachment-definition <name>
   ```

5. Optional: Confirm that the additional network CR was deleted by running the following command:
$ oc get network-attachment-definition --all-namespaces
8.1. ABOUT OPENSHIFT SDN

OpenShift Container Platform uses a software-defined networking (SDN) approach to provide a unified cluster network that enables communication between Pods across the OpenShift Container Platform cluster. 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. NetworkPolicy is the default mode in OpenShift Container Platform 4.2.

- 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.

8.2. ASSIGNING EGRESS IPS TO A PROJECT

As a cluster administrator, you can configure OpenShift Software Defined Network (SDN) to assign one or more egress IP addresses to a project. All outgoing external connections from the specified project will share the same, fixed source IP, allowing external resources to recognize the traffic based on the egress IP. An egress IP address assigned to a project is different from the egress router, which is used to send traffic to specific destinations.

Egress IPs 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.

**IMPORTANT**

Egress IPs 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. 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. This is the preferred approach.
In the manually assigned approach, a list of one or more egress IP address is assigned to a node. You set the egressIPs parameter of each node’s HostSubnet resource to indicate the IP addresses that can be hosted by a node.

Namespaces that request an egress IP addresses are matched with nodes that are able to host those egress IP addresses, and then the egress IP addresses are assigned to those nodes. If egressIPs 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 egress IP addresses is unreachable and there are nodes that are able to host those egress IP addresses, then the egress IP addresses will move to a new node. When the original egress IP address node comes back online, the egress IP addresses automatically move 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.

8.2.1. Enabling automatically assigned egress IPs 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.
- You must be logged in to the cluster with the cluster-admin role.

Procedure

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

   ```
   $ oc patch netnamespace <project_name> --type=merge -p '{
   "egressIPs": [
   "<ip_address>"
   ]}'
   ```

   1 Specify the name of the project.
   2 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:

   ```
   $ 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>"]
   }'
   
1. Specify a node name.
2. 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": ["192.168.1.0/24"]}'
   ```

3. 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.

### 8.2.2. Configuring manually assigned egress IPs

In OpenShift Container Platform you can associate one or more egress IPs with 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**

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

   ```bash
   $ oc patch netnamespace <project> --type=merge -p \\
   '{
   "egressIPs": [
   "<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:

```
$ 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:

```
$ oc patch hostsubnet <node_name> --type=merge -p \
    '{
        "egressIPs": [
            "<ip_address_1>",
            "<ip_address_N>"
        ]
    }'
```

1. Specify the name of the project.
2. 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.

### 8.3. USING MULTICAST

#### 8.3.1. 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 network plug-in, 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.

### 8.3.2. 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
  
  1 The `namespace` for the project you want to enable multicast for.
  ```

### 8.3.3. 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=
  
  1 The `namespace` for the project you want to disable multicast for.
  ```
8.4. 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.

8.4.1. 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:

   
   $ oc adm pod-network join-projects --to=<project1> <project2> <project3>

   Alternatively, instead of specifying specific project names, you can use the --selector=<project_selector> option to specify projects based upon an associated label.

2. Optional: Run the following command to view the pod networks that you have joined together:

   
   $ oc get netnamespaces

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

8.4.2. 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:

  ```bash
  $ 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.

### 8.4.3. 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:

  ```bash
  $ 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.

### 8.5. 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.

#### 8.5.1. 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.

#### 8.5.2. 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:

   ```
   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:

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

   The command returns output similar to the following example:

   ```
   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:

```
$ 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.

### 8.5.3. kube-proxy configuration parameters

You can modify the following **kubeProxyConfig** parameters:

<table>
<thead>
<tr>
<th>Table 8.1. Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
</tr>
<tr>
<td>------------</td>
</tr>
<tr>
<td><strong>iptablesSyncPeriod</strong></td>
</tr>
<tr>
<td><strong>proxyArguments.iptables-min-sync-period</strong></td>
</tr>
</tbody>
</table>
CHAPTER 9. CONFIGURING ROUTES

9.1. ROUTE CONFIGURATION

9.1.1. 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
   ```

9.1.2. 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:

  ```
  apiVersion: v1
  ```
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.

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

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.

9.1.3. 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.

  $ tcpdump -s 0 -i any -w /tmp/dump.pcap host <podip 1> && host <podip 2> ①

  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:

  $ 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.

9.1.4. Using cookies to keep route statefulness
OpenShift Container Platform provides sticky sessions, which enables stateful application traffic by ensuring all traffic hits the same endpoint. However, if the endpoint Pod terminates, whether through restart, scaling, or a change in configuration, this statefulness can disappear.

OpenShift Container Platform can use cookies to configure session persistence. The Ingress controller selects an endpoint to handle any user requests, and creates a cookie for the session. The cookie is passed back in the response to the request and the user sends the cookie back with the next request in the session. The cookie tells the Ingress Controller which endpoint is handling the session, ensuring that client requests use the cookie so that they are routed to the same Pod.

9.1.4.1. 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
   ```

9.2. 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.

9.2.1. 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:

```
$ openssl rsa -in password_protected_tls.key -out tls.key
```

**Procedure**

This procedure creates a Route resource with a custom certificate and reencryp 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 reencryp TLS termination and a custom certificate:

  ```
  $ oc create route reencrypt --service=frontend --cert=tls.crt --key=tls.key --dest-ca-cert=destca.crt --ca-cert=ca.crt --hostname=www.example.com
  ```

  If you examine the resulting Route resource, it should look similar to the following:

**YAML Definition of the Secure Route**

```yaml
apiVersion: v1
cr:
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-----

cacert:
  certificate: |
  -----BEGIN CERTIFICATE-----
  [...] 
  -----END CERTIFICATE-----
```

9.2.2. 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
```

See `oc create route reencrypt --help` for more options.
spec:
  host: www.example.com
to:
  kind: Service
  name: frontend
tls:
  termination: edge
  key: |
    -----BEGIN PRIVATE KEY-----
    [...]  
    -----END PRIVATE KEY-----
  certificate: |
    -----BEGIN CERTIFICATE-----
    [...]  
    -----END CERTIFICATE-----
caCertificate: |
    -----BEGIN CERTIFICATE-----
    [...]  
    -----END CERTIFICATE-----

See `oc create route edge --help` for more options.
CHAPTER 10. CONFIGURING INGRESS CLUSTER TRAFFIC

10.1. 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 <strong>NodePort</strong></td>
<td>Expose a service on all nodes in the cluster.</td>
</tr>
</tbody>
</table>

10.2. 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.

10.2.1. 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.

**10.2.2. 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.

**Procedure**

1. Log in to OpenShift Container Platform.

2. Create a new project for your service:

   ```
   $ oc new-project <project_name>
   ```

   For example:

   ```
   $ oc new-project myproject
   ```

3. 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/openshift3/mysql-55-rhel7

4. Run the following command to see that the new service is created:

$ oc get svc -n openshift-ingress
NAME                      TYPE           CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE
router-default            LoadBalancer   172.30.16.119   52.230.228.163 80:30745/TCP,443:32561/TCP   2d6h
router-internal-default   ClusterIP      172.30.101.15   <none>          80/TCP,443/TCP,1936/TCP      2d6h

By default, the new service does not have an external IP address.

10.2.3. Exposing the service by creating a route

You can expose the service as a route 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-55-rhel7
   route "mysql-55-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)]>

10.2.4. 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:

```
# 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`.

10.2.5. 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`.

10.2.6. 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.

10.3. 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.

10.3.1. 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.

10.3.2. 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.

Procedure

1. Log in to OpenShift Container Platform.

2. Create a new project for your service:

   `$ oc new-project <project_name>`

   For example:

   `$ oc new-project myproject`

3. 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/openshift3/mysql-55-rhel7`
4. Run the following command to see that the new service is created:

```
$ oc get svc -n openshift-ingress
NAME                      TYPE           CLUSTER-IP      EXTERNAL-IP      PORT(S)     AGE
router-default            LoadBalancer   172.30.16.119   52.230.228.163   80:30745/TCP,443:32561/TCP   2d6h
router-internal-default   ClusterIP      172.30.101.15   <none>           80/TCP,443/TCP,1936/TCP     2d6h
```

By default, the new service does not have an external IP address.

10.3.3. Exposing the service by creating a route

You can expose the service as a route 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-55-rhel7
   route "mysql-55-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.
   ```
10.3.4. 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:

   Example 10.1. 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
   ```

   - Enter a descriptive name for the load balancer service.
   - Enter the same port that the service you want to expose is listening on.
   - Enter `loadbalancer` as the type.
   - Enter the name of the service.

4. Save and exit the file.

5. Run the following command to create the service:

   ```bash
   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 -n openshift-ingress

<table>
<thead>
<tr>
<th>NAME</th>
<th>TYPE</th>
<th>CLUSTER-IP</th>
<th>EXTERNAL-IP</th>
<th>PORT(S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>router-default</td>
<td>LoadBalancer</td>
<td>172.30.16.119</td>
<td>52.230.228.163</td>
<td>80:30745/TCP,443:32561/TCP 2d6h</td>
</tr>
<tr>
<td>router-internal-default</td>
<td>ClusterIP</td>
<td>172.30.101.15</td>
<td>&lt;none&gt;</td>
<td>80/TCP,443/TCP,1936/TCP 2d6h</td>
</tr>
</tbody>
</table>

   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)]>

10.4. 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.

10.4.1. 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

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.

10.4.2. 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.

Procedure

1. Log in to OpenShift Container Platform.

2. Create a new project for your service:

   $ oc new-project <project_name>

   For example:

   $ oc new-project myproject

3. 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/openshift3/mysql-55-rhel7
4. Run the following command to see that the new service is created:

```
$ oc get svc -n openshift-ingress
NAME                      TYPE           CLUSTER-IP      EXTERNAL-IP      PORT(S)     AGE
router-default            LoadBalancer   172.30.16.119   52.230.228.163  80:30745/TCP,443:32561/TCP  2d6h
router-internal-default   ClusterIP      172.30.101.15   <none>          80/TCP,443/TCP,1936/TCP  2d6h
```

By default, the new service does not have an external IP address.

10.4.3. Exposing the service by creating a route

You can expose the service as a route 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-55-rhel7
   route "mysql-55-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.
   ```
10.5. 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**.

10.5.1. 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.

10.5.2. 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.
Procedure

1. Log in to OpenShift Container Platform.

2. Create a new project for your service:

   ```bash
   $ oc new-project <project_name>
   
   For example:
   ```

   ```bash
   $ oc new-project myproject
   ```

3. 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   -e MYSQL_DATABASE=mysqldb \n   registry.redhat.io/openshift3/mysql-55-rhel7
   ```

4. Run the following command to see that the new service is created:

   ```bash
   $ oc get svc -n openshift-ingress
   NAME                      TYPE           CLUSTER-IP      EXTERNAL-IP      PORT(S)                   AGE
   router-default            LoadBalancer   172.30.16.119   52.230.228.163   80:30745/TCP,443:32561/TCP   2d6h
   router-internal-default   ClusterIP      172.30.101.15   <none>           80/TCP,443/TCP,1936/TCP      2d6h
   ```

   By default, the new service does not have an external IP address.

10.5.3. Exposing the service by creating a route

You can expose the service as a route 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-55-rhel7
   route "mysql-55-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)]>
```
CHAPTER 11. 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 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.machineCIDR`, `networking.clusterNetwork.cidr`, and `networking.serviceNetwork` fields from your installation configuration.

11.1. 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
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: |
   <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:

   ```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
   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:
   ```

   1. httpProxy field.
   2. httpsProxy field.
   3. noProxy field.
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 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.machineCIDR 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.

5. 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.

11.2. 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:

   apiVersion: config.openshift.io/v1
   kind: Proxy
   metadata:
     name: cluster
   spec: {}
   status: {}
3. Save the file to apply the changes.
CHAPTER 12. 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.

12.1. 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.machineCIDR`, `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`.

   **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.

   **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.

   **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.

### 12.2. 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
class: 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: |
  -<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:

```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
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:
    - http://www.google.com
    - https://www.google.com
  trustedCA:
    name: user-ca-bundle
```

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 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.machineCIDR` 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.
5. 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.

---

**12.3. 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:

```yaml
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.