OpenShift Dedicated 4

Networking

Configuring OpenShift Dedicated networking
OpenShift Dedicated 4 Networking

Configuring OpenShift Dedicated networking
Abstract

This document provides information about networking for OpenShift Dedicated clusters.
# Table of Contents

## CHAPTER 1. INGRESS OPERATOR IN OPENSHIFT DEDICATED

1.1. OPENSHIFT DEDICATED INGRESS OPERATOR ............................................. 3  
1.2. VIEW THE DEFAULT INGRESS CONTROLLER .......................................... 3  
1.3. VIEW INGRESS OPERATOR STATUS ...................................................... 3  
1.4. VIEW INGRESS CONTROLLER LOGS ...................................................... 3  
1.5. VIEW INGRESS CONTROLLER STATUS .................................................. 4  

## CHAPTER 2. OPENSHIFT SDN DEFAULT CNI NETWORK PROVIDER

2.1. ENABLING MULTICAST FOR A PROJECT .................................................. 5  
   2.1.1. About multicast ............................................................................... 5  
   2.1.2. Enabling multicast between pods .................................................... 5  

## CHAPTER 3. CONFIGURING A CLUSTER-WIDE PROXY DURING INSTALLATION

3.1. PREREQUISITES FOR CONFIGURING A CLUSTER-WIDE PROXY ............... 8  
   3.1.1. General requirements ...................................................................... 8  
   3.1.2. Network requirements ..................................................................... 8  
3.2. CONFIGURING A CLUSTER-WIDE PROXY DURING INSTALLATION .......... 10  
3.3. CONFIGURING OR UPDATING YOUR CLUSTER-WIDE PROXY AFTER INSTALLATION ......................................................... 11  
   3.3.1. Responsibilities for additional trust bundles ........................................ 12  

## CHAPTER 4. CIDR RANGE DEFINITIONS

4.1. MACHINE CIDR ....................................................................................... 13  
4.2. SERVICE CIDR ...................................................................................... 13  
4.3. POD CIDR ............................................................................................... 13  
4.4. HOST PREFIX ............................................................................................ 13  

## CHAPTER 5. NETWORK POLICY

5.1. ABOUT NETWORK POLICY ...................................................................... 14  
   5.1.1. About network policy ...................................................................... 14  
   5.1.2. Optimizations for network policy ...................................................... 16  
   5.1.3. Next steps ....................................................................................... 16  
5.2. CREATING A NETWORK POLICY ......................................................... 16  
   5.2.1. Example NetworkPolicy object ....................................................... 17  
   5.2.2. Creating a network policy using the CLI ......................................... 17  
   5.2.3. Creating a network policy using OpenShift Cluster Manager .......... 19  
5.3. VIEWING A NETWORK POLICY .............................................................. 20  
   5.3.1. Example NetworkPolicy object ....................................................... 20  
   5.3.2. Viewing network policies using the CLI .......................................... 21  
   5.3.3. Viewing network policies using OpenShift Cluster Manager .......... 22  
5.4. DELETING A NETWORK POLICY ........................................................... 23  
   5.4.1. Deleting a network policy using the CLI .......................................... 23  
   5.4.2. Deleting a network policy using OpenShift Cluster Manager .......... 24  
5.5. CONFIGURING MULTITENANT ISOLATION WITH NETWORK POLICY ................. 24  
   5.5.1. Configuring multitenant isolation by using network policy ............... 24
1.1. OPENSHIFT DEDICATED INGRESS OPERATOR

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

The Ingress Operator makes it possible for external clients to access your service by deploying and managing one or more HAProxy-based Ingress Controllers to handle routing. Red Hat Site Reliability Engineers (SRE) manage the Ingress Operator for OpenShift Dedicated clusters. While you cannot alter the settings for the Ingress Operator, you may view the default Ingress Controller configurations, status, and logs as well as the Ingress Operator status.

1.2. VIEW THE DEFAULT INGRESS CONTROLLER

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

Every new OpenShift Dedicated 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

1.3. 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

1.4. VIEW INGRESS CONTROLLER LOGS

You can view your Ingress Controller logs.

Procedure

- View your Ingress Controller logs:

  $ oc logs --namespace=openshift-ingress-operator deployments/ingress-operator -c <container_name>
1.5. 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>
  ```
2.1. Enabling Multicast for a Project

2.1.1. 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 Dedicated pods is disabled by default. If you are using the OpenShift SDN default Container Network Interface (CNI) network provider, you can enable multicast on a per-project basis.

When using the OpenShift SDN network plug-in in **networkpolicy** isolation mode:

- Multicast packets sent by a pod will be delivered to all other pods in the project, regardless of **NetworkPolicy** objects. Pods might be able to communicate over multicast even when they cannot communicate over unicast.

- Multicast packets sent by a pod in one project will never be delivered to pods in any other project, even if there are **NetworkPolicy** objects that allow communication between the projects.

When using the OpenShift SDN network plug-in in **multitenant** isolation mode:

- Multicast packets sent by a pod will be delivered to all other pods in the project.

- Multicast packets sent by a pod in one project will be delivered to pods in other projects only if each project is joined together and multicast is enabled in each joined project.

2.1.2. Enabling multicast between pods

You can enable multicast between pods for your project.

**Prerequisites**

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

- You must log in to the cluster with a user that has the **cluster-admin** or the **dedicated-admin** role.

**Procedure**

- Run the following command to enable multicast for a project. Replace `<namespace>` with the namespace for the project you want to enable multicast for.

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

To verify that multicast is enabled for a project, complete the following procedure:

1. Change your current project to the project that you enabled multicast for. Replace `<project>` with the project name.

```
$ oc project <project>
```

2. Create a pod to act as a multicast receiver:

```
$ cat <<EOF| oc create -f -
apiversion: v1
kind: Pod
metadata:
  name: mlistener
  labels:
    app: multicast-verify
spec:
  containers:
    - name: mlistener
      image: registry.access.redhat.com/ubi8
      command: ["/bin/sh", "-c"]
      args:
        ["dnf -y install socat hostname && sleep inf"]
      ports:
        - containerPort: 30102
          name: mlistener
          protocol: UDP
EOF
```

3. Create a pod to act as a multicast sender:

```
$ cat <<EOF| oc create -f -
apiversion: v1
kind: Pod
metadata:
  name: msender
  labels:
    app: multicast-verify
spec:
  containers:
    - name: msender
      image: registry.access.redhat.com/ubi8
      command: ["/bin/sh", "-c"]
      args:
        ["dnf -y install socat && sleep inf"]
EOF
```

4. In a new terminal window or tab, start the multicast listener.
   a. Get the IP address for the Pod:

```
$ POD_IP=$(oc get pods mlistener -o jsonpath='{.status.podIP}')
```
b. Start the multicast listener by entering the following command:

```
$ oc exec mlistener -i -t --
   socat UDP4-RECVFROM:30102,ip-add-membership=224.1.0.1:$POD_IP,fork
   EXEC:hostname
```

5. Start the multicast transmitter.

a. Get the pod network IP address range:

```
$ CIDR=$(oc get Network.config.openshift.io cluster
   -o jsonpath='{.status.clusterNetwork[0].cidr}"
```

b. To send a multicast message, enter the following command:

```
$ oc exec msender -i -t --
   /bin/bash -c "echo | socat STDIO UDP4-DATAGRAM:224.1.0.1:30102,range=$CIDR,ip-multicast-ttl=64"
```

If multicast is working, the previous command returns the following output:

```
mlistener
```
CHAPTER 3. CONFIGURING A CLUSTER-WIDE PROXY DURING INSTALLATION

You can configure a cluster-wide proxy during cluster installation or after the cluster has been installed.

If you use a cluster-wide proxy, you are responsible for the following:

- Maintaining the availability of the proxy to the cluster.
- Understanding that if the proxy becomes unavailable, then it may impact the health and supportability of the cluster.

IMPORTANT

Cluster-wide proxy is a functionally-complete feature and suitable for production workloads. There are additional considerations that need to be added to documentation, and until then, this feature is considered a Technology Preview.

3.1. PREREQUISITES FOR CONFIGURING A CLUSTER-WIDE PROXY

To configure a cluster-wide proxy, you must meet the following requirements. These requirements are valid for both fresh installation and post installation proxy configuration.

3.1.1. General requirements

- You are the cluster owner.
- Your account has sufficient privileges.
- You have added the `ec2.<region>.amazonaws.com`, `elasticloadbalancing.<region>.amazonaws.com`, and `s3.<region>.amazonaws.com` endpoints to your virtual private cloud (VPC) endpoint. These endpoints are required to complete requests from the nodes to the AWS EC2 API. Because the proxy works on the container level, not the node level, you must route these requests to the AWS EC2 API through the AWS private network. Adding the public IP address of the EC2 API to your allowlist in your proxy server is not sufficient.
- You must have a Customer Cloud Subscription (CCS) cluster with a VPC that the proxy can access.
- You have the `ocm` CLI installed and configured.

3.1.2. Network requirements

- If your proxy re-encrypts egress traffic, you must create exclusions to the domain and port combinations. The following table offers guidance into these exceptions.
  - Allowlist the following OpenShift URLs for re-encryption.
<table>
<thead>
<tr>
<th>Address</th>
<th>Protocol/Port</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>sso.redhat.com</td>
<td>http/443</td>
<td>The <a href="https://cloud.redhat.com/openshift">https://cloud.redhat.com/openshift</a> site uses authentication from sso.redhat.com to download the cluster pull secret and use Red Hat SaaS solutions to facilitate monitoring of your subscriptions, cluster inventory, and chargeback reporting.</td>
</tr>
</tbody>
</table>

- Allowlist the following site reliability engineering (SRE) and management URLs for re-encryption.

<table>
<thead>
<tr>
<th>Address</th>
<th>Protocol/Port</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>*.osdsecuritylogs.splunkcloud.com</td>
<td>tcp/99 97</td>
<td>Used by the splunk-forwarder-operator as a log forwarding endpoint to be used by Red Hat SRE for log-based alerting.</td>
</tr>
</tbody>
</table>
Additional Resources

For more information, see Getting started with OpenShift Dedicated for a basic cluster installation workflow.

**IMPORTANT**

The use of a proxy server to perform TLS re-encryption is currently not supported if the server is acting as a transparent forward proxy where it is not configured on-cluster via the `--http-proxy` or `--https-proxy` arguments.

A transparent forward proxy intercepts the cluster’s traffic, but it is not actually configured on the cluster itself.

### 3.2. CONFIGURING A CLUSTER-WIDE PROXY DURING INSTALLATION

You can add a proxy during cluster installation. Prior to installation, however, you should verify that the proxy is accessible from the intended cluster virtual private cloud (VPC) and its private subnets.

**WARNING**

Only cluster system egress traffic is proxied, including calls to the AWS API. A system-wide proxy does not affect user workloads. It only affects system components.

**Procedure**

- To create a cluster with a proxy, run the following command:

  ```bash
  $ ocm create cluster <other_arguments_here> 
  --additional-trust-bundle-file <path_to_CA_bundle_file> 
  --http-proxy http://<username>:<pswd>@<ip>:<port> 
  --https-proxy http(s)://<username>:<pswd>@<ip>:<port>
  ```

  1 The `http-proxy`, `https-proxy`, and `additional-trust-bundle-file` arguments are all optional.
If you use the `additional-trust-bundle-file` option without an `http(s)-proxy` argument, the passed additional trust bundle is set on the cluster, but it is not configured to be used with

The `additional-trust-bundle-file` argument is a file path pointing to a bundle of PEM-encoded X.509 certificates, which are all concatenated together. The `additionalTrustBundle` parameter is required unless the proxy’s identity certificate is signed by an authority from the RHCOS trust bundle. If you use an MITM transparent proxy network that does not require additional proxy configuration but requires additional CAs, you must provide the MITM CA certificate.

The `http-proxy` and `https-proxy` arguments must point to a valid URL.

### 3.3. CONFIGURING OR UPDATING YOUR CLUSTER-WIDE PROXY AFTER INSTALLATION

As the cluster owner, you may wish to add a proxy to your created cluster after installation, or you may wish to make changes to your proxy that you configured during cluster installation. The `ocm` CLI provides some options for adding a proxy to your cluster or modifying an existing proxy on your cluster.

You may need to perform these actions if:

- the cluster-wide proxy is configured after installation,
- the proxy’s network address needs to be updated, and/or
- any of the proxy’s certificate authorities have expired and the additional trust bundle needs to be replaced.

**NOTE**

The cluster applies the configuration to the cluster’s control plane and worker nodes. This process results in each node in the cluster temporarily being placed into an unschedulable state and drained of its workloads while applying the configuration. Each node will be restarted as part of this process.

**Procedure**

- To edit a cluster, run the following command:

  ```bash
  $ ocm edit cluster \
  --cluster $CLUSTER_NAME \
  --additional-trust-bundle-file $CA_BUNDLE_FILE \
  --http-proxy $HTTP_PROXY \
  --https-proxy $HTTPS_PROXY
  ```

  While the `additional-trust-bundle-file`, `http-proxy`, and `https-proxy` arguments are optional, if you set a `additional-trust-bundle-file` without either an `http-proxy` or `https-proxy` argument, then the additional trust bundle will still be used for verifying cluster system egress traffic.

- You can verify that the proxy and certificate authority configuration updates have been successfully applied to your cluster by:
  - All of the MachineConfigPools are updated. Run the following command to see their status:
$ oc get machineconfigpools

Sample Output

<table>
<thead>
<tr>
<th>NAME</th>
<th>CONFIG</th>
<th>UPDATED</th>
<th>UPDATING</th>
<th>DEGRADED</th>
</tr>
</thead>
<tbody>
<tr>
<td>master</td>
<td>rendered-master-d9a03f612a432095dcde6dcf44597d90</td>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>worker</td>
<td>rendered-worker-f6827a4efe21e155c25c21b43c46f65e</td>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>

- As the cluster owner, the following command displays the proxy status:

$ oc get proxy cluster -o yaml

Sample Output

```yaml
apiVersion: config.openshift.io/v1
kind: Proxy
spec:
  httpProxy: http://proxy.host.domain:<port>
  httpsProxy: https://proxy.host.domain:<port>
  <...more...>
status:
  httpProxy: http://proxy.host.domain:<port>
  httpsProxy: https://proxy.host.domain:<port>
  <...more...>
```

NOTE

You should not attempt to change the proxy or additional trust bundle configuration on the cluster itself. These changes should always be done via the `ocm` command-line tools. Any changes that are made directly to the cluster will be reverted automatically.

3.3.1. Responsibilities for additional trust bundles

If you supplied an additional trust bundle file, you are responsible for the following:

- Ensuring that the contents of the additional trust bundle are valid,
- Ensuring that the certificates, including intermediary certificates, contained in the additional trust bundle have not expired, and
- Tracking the expiry and performing any necessary renewals for certificates contained in the additional trust bundle, and subsequently updating the cluster’s configuration with the updated additional trust bundle.
CHAPTER 4. CIDR RANGE DEFINITIONS

You must specify non-overlapping ranges for the following CIDR ranges.

NOTE

Machine CIDR ranges cannot be changed after creating your cluster.

4.1. MACHINE CIDR

In the Machine CIDR field, you must specify the IP address range for machines or cluster nodes. This range must encompass all CIDR address ranges for your virtual private cloud (VPC) subnets. Subnets must be contiguous. A minimum IP address range of 128 addresses, using the subnet prefix /25, is supported for single availability zone deployments. A minimum address range of 256 addresses, using the subnet prefix /24, is supported for deployments that use multiple availability zones. The default is 10.0.0.0/16. This range must not conflict with any connected networks.

4.2. SERVICE CIDR

In the Service CIDR field, you must specify the IP address range for services. The range must be large enough to accommodate your workload. The address block must not overlap with any external service accessed from within the cluster. The default is 172.30.0.0/16. This address block needs to be the same between clusters.

4.3. POD CIDR

In the pod CIDR field, you must specify the IP address range for pods. The range must be large enough to accommodate your workload. The address block must not overlap with any external service accessed from within the cluster. The default is 10.128.0.0/14. This address block needs to be the same between clusters.

4.4. HOST PREFIX

In the Host Prefix field, you must specify the subnet prefix length assigned to pods scheduled to individual machines. The host prefix determines the pod IP address pool for each machine. For example, if the host prefix is set to /23, each machine is assigned a /23 subnet from the pod CIDR address range. The default is /23, allowing 512 cluster nodes, and 512 pods per node (both of which are beyond our maximum supported).
CHAPTER 5. NETWORK POLICY

5.1. ABOUT NETWORK POLICY

As a cluster administrator, you can define network policies that restrict traffic to pods in your cluster.

5.1.1. About network policy

In a cluster using a Kubernetes Container Network Interface (CNI) plug-in that supports Kubernetes network policy, network isolation is controlled entirely by `NetworkPolicy` objects. In OpenShift Dedicated 4, OpenShift SDN supports using network policy in its default network isolation mode.

**WARNING**

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

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 Dedicated Ingress Controller:**
  To make a project allow only connections from the OpenShift Dedicated Ingress Controller, add the following `NetworkPolicy` object.

  ```yaml
  apiVersion: networking.k8s.io/v1
  kind: NetworkPolicy
  metadata:
    name: allow-from-openshift-ingress
  spec:
  ```
Only accept connections from pods within a project:
To make pods accept connections from other pods in the same project, but reject all other connections from pods in other projects, add the following `NetworkPolicy` object:

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

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

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

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

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

5.1.2. Optimizations for network policy

Use a network policy to isolate pods that are differentiated from one another by labels within a namespace.

NOTE

The guidelines for efficient use of network policy rules applies to only the OpenShift SDN cluster network provider.

It is inefficient to apply NetworkPolicy objects to large numbers of individual pods in a single namespace. Pod labels do not exist at the IP address level, so a network policy generates a separate Open vSwitch (OVS) flow rule for every possible link between every pod selected with a podSelector.

For example, if the spec podSelector and the ingress podSelector within a NetworkPolicy object each match 200 pods, then 40,000 (200*200) OVS flow rules are generated. This might slow down a node.

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

- Reduce the number of OVS flow rules by using namespaces to contain groups of pods that need to be isolated. NetworkPolicy objects that select a whole namespace, by using the namespaceSelector or an empty podSelector, generate only a single OVS flow rule that matches the VXLAN virtual network ID (VNID) of the namespace.

- Keep the pods that do not need to be isolated in their original namespace, and move the pods that require isolation into one or more different namespaces.

- Create additional targeted cross-namespace network policies to allow the specific traffic that you do want to allow from the isolated pods.

5.1.3. Next steps

- Creating a network policy

5.2. CREATING A NETWORK POLICY

```yaml
ingress:
  - from:
      - namespaceSelector:
          matchLabels:
            project: project_name
      podSelector:
        matchLabels:
          name: test-pods
```
As a user with the **admin** role, you can create a network policy for a namespace.

### 5.2.1. Example NetworkPolicy object

The following annotates an example NetworkPolicy object:

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

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

### 5.2.2. Creating a network policy using the CLI

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

**NOTE**

If you log in with a user with the **cluster-admin** role, then you can create a network policy in any namespace in the cluster.

**Prerequisites**

- Your cluster uses a cluster network provider that supports **NetworkPolicy** objects, such as the OpenShift SDN network provider with `mode: NetworkPolicy` set. This mode is the default for OpenShift SDN.
- You installed the OpenShift CLI (```
  oc```).
- You are logged in to the cluster with a user with **admin** privileges.
You are working in the namespace that the network policy applies to.

Procedure

1. Create a policy rule:
   a. Create a `<policy_name>.yaml` file:
      ```
      $ touch <policy_name>.yaml
      ```
      where:
      ```
      <policy_name>
      Specifies the network policy file name.
      ```
   b. Define a network policy in the file that you just created, such as in the following examples:
      ```yaml
      Kind: NetworkPolicy
      apiVersion: networking.k8s.io/v1
      metadata:
      name: deny-by-default
      spec:
      podSelector:
      ingress: []
      ```
      ```yaml
      Kind: NetworkPolicy
      apiVersion: networking.k8s.io/v1
      metadata:
      name: allow-same-namespace
      spec:
      podSelector:
      ingress:
      - from:
        - podSelector: {}
      ```
2. To create the network policy object, enter the following command:
   ```
   $ oc apply -f <policy_name>.yaml -n <namespace>
   ```
   where:
   ```
   <policy_name>
   Specifies the network policy file name.
   <namespace>
   Optional: Specifies the namespace if the object is defined in a different namespace than the current namespace.
   ```
   Example output
NOTE
If you log in to the web console with `cluster-admin` privileges, you have a choice of creating a network policy in any namespace in the cluster directly in YAML or from a form in the web console.

5.2.3. Creating a network policy using OpenShift Cluster Manager

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

**Prerequisites**

- You logged in to OpenShift Cluster Manager.
- You created an OpenShift Dedicated cluster.
- You configured an identity provider for your cluster.
- You added your user account to the configured identity provider.
- You created a project within your OpenShift Dedicated cluster.

**Procedure**

1. From OpenShift Cluster Manager, click on the cluster you want to access.
2. Click Open console to navigate to the OpenShift web console.
3. Click on your identity provider and provide your credentials to log in to the cluster.
4. From the administrator perspective, under Networking, click NetworkPolicies.
5. Click Create NetworkPolicy.
6. Provide a name for the policy in the Policy name field.
7. Optional: You can provide the label and selector for a specific pod if this policy applies only to one or more specific pods. If you do not select a specific pod, then this policy will be applicable to all pods on the cluster.
8. Optional: You can block all ingress and egress traffic by using the Deny all ingress traffic or Deny all egress traffic checkboxes.
9. You can also add any combination of ingress and egress rules, allowing you to specify the port, namespace, or IP blocks you want to approve.
10. Add ingress rules to your policy:
    a. Select Add ingress rule to configure a new rule. This action creates a new Ingress rule row with an Add allowed source drop-down menu that enables you to specify how you want to limit inbound traffic. The drop-down menu offers three options to limit your ingress traffic:
        - Allow pods from the same namespace limits traffic to pods within the same
namespace. You can specify the pods in a namespace, but leaving this option blank allows all of the traffic from pods in the namespace.

- **Allow pods from inside the cluster** limits traffic to pods within the same cluster as the policy. You can specify namespaces and pods from which you want to allow inbound traffic. Leaving this option blank allows inbound traffic from all namespaces and pods within this cluster.

- **Allow peers by IP block** limits traffic from a specified Classless Inter-Domain Routing (CIDR) IP block. You can block certain IPs with the exceptions option. Leaving the CIDR field blank allows all inbound traffic from all external sources.

b. You can restrict all of your inbound traffic to a port. If you do not add any ports then all ports are accessible to traffic.

11. **Add egress rules to your network policy:**

a. Select **Add egress rule** to configure a new rule. This action creates a new **Egress rule** row with an **Add allowed destination** drop-down menu that enables you to specify how you want to limit outbound traffic. The drop-down menu offers three options to limit your egress traffic:

- **Allow pods from the same namespace** limits outbound traffic to pods within the same namespace. You can specify the pods in a namespace, but leaving this option blank allows all of the traffic from pods in the namespace.

- **Allow pods from inside the cluster** limits traffic to pods within the same cluster as the policy. You can specify namespaces and pods from which you want to allow outbound traffic. Leaving this option blank allows outbound traffic from all namespaces and pods within this cluster.

- **Allow peers by IP block** limits traffic from a specified CIDR IP block. You can block certain IPs with the exceptions option. Leaving the CIDR field blank allows all outbound traffic from all external sources.

b. You can restrict all of your outbound traffic to a port. If you do not add any ports then all ports are accessible to traffic.

### 5.3. VIEWING A NETWORK POLICY

As a user with the **admin** role, you can view a network policy for a namespace.

#### 5.3.1. Example NetworkPolicy object

The following annotates an example NetworkPolicy object:

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

---

1. replace the placeholder with the actual name of the network policy.
2. replace the placeholder with the actual pod selector.
The name of the NetworkPolicy object.

A selector that describes the pods to which the policy applies. The policy object can only select pods in the project that defines the NetworkPolicy object.

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

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

### 5.3.2. Viewing network policies using the CLI

You can examine the network policies in a namespace.

**NOTE**

If you log in with a user with the `cluster-admin` role, then you can view any network policy in the cluster.

**Prerequisites**

- You installed the OpenShift CLI (`oc`).
- You are logged in to the cluster with a user with `admin` privileges.
- You are working in the namespace where the network policy exists.

**Procedure**

- List network policies in a namespace:
  - To view network policy objects defined in a namespace, enter the following command:
    
    $ oc get networkpolicy
  
  - Optional: To examine a specific network policy, enter the following command:
    
    $ oc describe networkpolicy <policy_name> -n <namespace>

    where:

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

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

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

**Output for `oc describe` command**

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

**NOTE**
If you log in to the web console with `cluster-admin` privileges, you have a choice of viewing a network policy in any namespace in the cluster directly in YAML or from a form in the web console.

### 5.3.3. Viewing network policies using OpenShift Cluster Manager

You can view the configuration details of your network policy in Red Hat OpenShift Cluster Manager.

**Prerequisites**

- You logged in to OpenShift Cluster Manager.
- You created an OpenShift Dedicated cluster.
- You configured an identity provider for your cluster.
- You added your user account to the configured identity provider.
- You created a network policy.

**Procedure**

1. From the Administrator perspective in the OpenShift Cluster Manager web console, under Networking, click NetworkPolicies.
2. Select the desired network policy to view.
3. In the Network Policy details page, you can view all of the associated ingress and egress rules.
4. Select YAML on the network policy details to view the policy configuration in YAML format.
5.4. DELETING A NETWORK POLICY

As a user with the admin role, you can delete a network policy from a namespace.

5.4.1. Deleting a network policy using the CLI

You can delete a network policy in a namespace.

NOTE

If you log in with a user with the cluster-admin role, then you can delete any network policy in the cluster.

Prerequisites

- Your cluster uses a cluster network provider that supports NetworkPolicy objects, such as the OpenShift SDN network provider with mode: NetworkPolicy set. This mode is the default for OpenShift SDN.
- You installed the OpenShift CLI (oc).
- You are logged in to the cluster with a user with admin privileges.
- You are working in the namespace where the network policy exists.

Procedure

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

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

  where:

  <policy_name>
  Specifies the name of the network policy.

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

Example output

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

NOTE

If you log in to the web console with cluster-admin privileges, you have a choice of deleting a network policy in any namespace in the cluster directly in YAML or from the policy in the web console through the Actions menu.
5.4.2. Deleting a network policy using OpenShift Cluster Manager

You can delete a network policy in a namespace.

Prerequisites

- You logged in to OpenShift Cluster Manager.
- You created an OpenShift Dedicated cluster.
- You configured an identity provider for your cluster.
- You added your user account to the configured identity provider.

Procedure

1. From the Administrator perspective in the OpenShift Cluster Manager web console, under Networking, click NetworkPolicies.

2. Use one of the following methods for deleting your network policy:
   - Delete the policy from the Network Policies table:
     a. From the Network Policies table, select the stack menu on the row of the network policy you want to delete and then, click Delete NetworkPolicy.
   - Delete the policy using the Actions drop-down menu from the individual network policy details:
     a. Click on Actions drop-down menu for your network policy.
     b. Select Delete NetworkPolicy from the menu.

5.5. CONFIGURING MULTITENANT ISOLATION WITH NETWORK POLICY

As a cluster administrator, you can configure your network policies to provide multitenant network isolation.

NOTE

If you are using the OpenShift SDN cluster network provider, configuring network policies as described in this section provides network isolation similar to multitenant mode but with network policy mode set.

5.5.1. Configuring multitenant isolation by using network policy

You can configure your project to isolate it from pods and services in other project namespaces.

Prerequisites

- Your cluster uses a cluster network provider that supports NetworkPolicy objects, such as the OpenShift SDN network provider with mode: NetworkPolicy set. This mode is the default for OpenShift SDN.
You installed the OpenShift CLI (oc).

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

**Procedure**

1. Create the following NetworkPolicy objects:
   a. A policy named **allow-from-openshift-ingress**:
      
      ```bash
      $ cat << EOF | oc create -f -
      apiVersion: networking.k8s.io/v1
      kind: NetworkPolicy
      metadata:
        name: allow-from-openshift-ingress
      spec:
        ingress:
        - from:
          - namespaceSelector:
            matchLabels:
              policy-group.network.openshift.io/ingress: ""
        podSelector: {}
      policyTypes:
        - Ingress
      EOF
      ```

      **NOTE**

      policy-group.network.openshift.io/ingress: "" is the preferred namespace selector label for OpenShift SDN. You can use the network.openshift.io/policy-group: ingress namespace selector label, but this is a legacy label.

   b. A policy named **allow-from-openshift-monitoring**:
      
      ```bash
      $ cat << EOF | oc create -f -
      apiVersion: networking.k8s.io/v1
      kind: NetworkPolicy
      metadata:
        name: allow-from-openshift-monitoring
      spec:
        ingress:
        - from:
          - namespaceSelector:
            matchLabels:
              network.openshift.io/policy-group: monitoring
        podSelector: {}
      policyTypes:
        - Ingress
      EOF
      ```

   c. A policy named **allow-same-namespace**:
      
      ```bash
      $ cat << EOF | oc create -f -
      EOF
      ```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-same-namespace
spec:
podSelector:
ingress:
  - from:
    - podSelector: {}
EOF

2. Optional: To confirm that the network policies exist in your current project, enter the following command:

```
$ oc describe networkpolicy
```

Example output

<table>
<thead>
<tr>
<th>Name</th>
<th>allow-from-openshift-ingress</th>
</tr>
</thead>
<tbody>
<tr>
<td>Namespace</td>
<td>example1</td>
</tr>
<tr>
<td>Created on</td>
<td>2020-06-09 00:28:17 -0400 EDT</td>
</tr>
<tr>
<td>Labels</td>
<td>&lt;none&gt;</td>
</tr>
<tr>
<td>Annotations</td>
<td>&lt;none&gt;</td>
</tr>
<tr>
<td>Spec:</td>
<td>PodSelector: &lt;none&gt; (Allowing the specific traffic to all pods in this namespace)</td>
</tr>
<tr>
<td></td>
<td>Allowing ingress traffic:</td>
</tr>
<tr>
<td></td>
<td>To Port: &lt;any&gt; (traffic allowed to all ports)</td>
</tr>
<tr>
<td></td>
<td>From:</td>
</tr>
<tr>
<td></td>
<td>NamespaceSelector: network.openshift.io/policy-group: ingress</td>
</tr>
<tr>
<td></td>
<td>Not affecting egress traffic</td>
</tr>
<tr>
<td></td>
<td>Policy Types: Ingress</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>allow-from-openshift-monitoring</th>
</tr>
</thead>
<tbody>
<tr>
<td>Namespace</td>
<td>example1</td>
</tr>
<tr>
<td>Created on</td>
<td>2020-06-09 00:29:57 -0400 EDT</td>
</tr>
<tr>
<td>Labels</td>
<td>&lt;none&gt;</td>
</tr>
<tr>
<td>Annotations</td>
<td>&lt;none&gt;</td>
</tr>
<tr>
<td>Spec:</td>
<td>PodSelector: &lt;none&gt; (Allowing the specific traffic to all pods in this namespace)</td>
</tr>
<tr>
<td></td>
<td>Allowing ingress traffic:</td>
</tr>
<tr>
<td></td>
<td>To Port: &lt;any&gt; (traffic allowed to all ports)</td>
</tr>
<tr>
<td></td>
<td>From:</td>
</tr>
<tr>
<td></td>
<td>NamespaceSelector: network.openshift.io/policy-group: monitoring</td>
</tr>
<tr>
<td></td>
<td>Not affecting egress traffic</td>
</tr>
<tr>
<td></td>
<td>Policy Types: Ingress</td>
</tr>
</tbody>
</table>