



OpenShift Container Platform 4.3

Networking

Configuring and managing cluster networking

OpenShift Container Platform 4.3 Networking

Configuring and managing cluster networking

Legal Notice

Copyright © 2020 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

XFS[®] is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack[®] Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

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	7
1.1. OPENSIFT CONTAINER PLATFORM DNS	7
CHAPTER 2. ACCESSING HOSTS	8
2.1. ACCESSING HOSTS ON AMAZON WEB SERVICES IN AN INSTALLER-PROVISIONED INFRASTRUCTURE CLUSTER	8
CHAPTER 3. CLUSTER NETWORK OPERATOR IN OPENSIFT CONTAINER PLATFORM	9
3.1. CLUSTER NETWORK OPERATOR	9
3.2. VIEWING THE CLUSTER NETWORK CONFIGURATION	9
3.3. VIEWING CLUSTER NETWORK OPERATOR STATUS	10
3.4. VIEWING CLUSTER NETWORK OPERATOR LOGS	10
3.5. CLUSTER NETWORK OPERATOR CONFIGURATION	11
3.5.1. Configuration parameters for the OpenShift SDN network provider	12
3.5.2. Configuration parameters for the OVN-Kubernetes network provider	12
3.5.3. Cluster Network Operator example configuration	13
CHAPTER 4. DNS OPERATOR IN OPENSIFT CONTAINER PLATFORM	14
4.1. DNS OPERATOR	14
4.2. VIEW THE DEFAULT DNS	14
4.3. USING DNS FORWARDING	15
4.4. DNS OPERATOR STATUS	17
4.5. DNS OPERATOR LOGS	17
CHAPTER 5. INGRESS OPERATOR IN OPENSIFT CONTAINER PLATFORM	18
5.1. THE INGRESS CONFIGURATION ASSET	18
5.2. INGRESS CONTROLLER CONFIGURATION PARAMETERS	18
5.2.1. Ingress controller TLS profiles	21
5.2.2. Ingress controller endpoint publishing strategy	22
5.3. VIEW THE DEFAULT INGRESS CONTROLLER	22
5.4. VIEW INGRESS OPERATOR STATUS	23
5.5. VIEW INGRESS CONTROLLER LOGS	23
5.6. VIEW INGRESS CONTROLLER STATUS	23
5.7. SETTING A CUSTOM DEFAULT CERTIFICATE	23
5.8. SCALING AN INGRESS CONTROLLER	24
5.9. CONFIGURING INGRESS CONTROLLER SHARDING BY USING ROUTE LABELS	25
5.10. CONFIGURING INGRESS CONTROLLER SHARDING BY USING NAMESPACE LABELS	26
5.11. CONFIGURING AN INGRESS CONTROLLER TO USE AN INTERNAL LOAD BALANCER	27
5.12. CONFIGURING THE DEFAULT INGRESS CONTROLLER FOR YOUR CLUSTER TO BE INTERNAL	28
5.13. ADDITIONAL RESOURCES	29
CHAPTER 6. CONFIGURING NETWORK POLICY WITH OPENSIFT SDN	30
6.1. ABOUT NETWORK POLICY	30
6.2. EXAMPLE NETWORKPOLICY OBJECT	32
6.3. CREATING A NETWORKPOLICY OBJECT	33
6.4. DELETING A NETWORKPOLICY OBJECT	33
6.5. VIEWING NETWORKPOLICY OBJECTS	34
6.6. CONFIGURING MULTITENANT ISOLATION USING NETWORKPOLICY	34
6.7. CREATING DEFAULT NETWORK POLICIES FOR A NEW PROJECT	36
6.7.1. Modifying the template for new projects	36
6.7.2. Adding network policy objects to the new project template	37
CHAPTER 7. MULTIPLE NETWORKS	39

7.1. UNDERSTANDING MULTIPLE NETWORKS	39
7.1.1. Usage scenarios for an additional network	39
7.1.2. Additional networks in OpenShift Container Platform	39
7.2. ATTACHING A POD TO AN ADDITIONAL NETWORK	40
7.2.1. Adding a Pod to an additional network	40
7.2.1.1. Specifying Pod-specific addressing and routing options	41
7.3. REMOVING A POD FROM AN ADDITIONAL NETWORK	45
7.3.1. Removing a Pod from an additional network	45
7.4. CONFIGURING A BRIDGE NETWORK	46
7.4.1. Creating an additional network attachment with the bridge CNI plug-in	46
7.4.1.1. Configuration for bridge	47
7.4.1.1.1. bridge configuration example	49
7.4.1.2. Configuration for ipam CNI plug-in	49
7.4.1.2.1. Static IP address assignment configuration example	51
7.4.1.2.2. Dynamic IP address assignment configuration example	51
7.5. CONFIGURING A MACVLAN NETWORK	51
7.5.1. Creating an additional network attachment with the macvlan CNI plug-in	51
7.5.1.1. Configuration for macvlan CNI plug-in	52
7.5.1.1.1. macvlan configuration example	53
7.5.1.2. Configuration for ipam CNI plug-in	53
7.5.1.2.1. Static ipam configuration YAML	54
7.5.1.2.2. Dynamic ipam configuration YAML	55
7.5.1.2.3. Static IP address assignment configuration example	55
7.5.1.2.4. Dynamic IP address assignment configuration example	55
7.6. CONFIGURING AN IPVLAN NETWORK	55
7.6.1. Creating an additional network attachment with the ipvlan CNI plug-in	56
7.6.1.1. Configuration for ipvlan	57
7.6.1.1.1. ipvlan configuration example	58
7.6.1.2. Configuration for ipam CNI plug-in	58
7.6.1.2.1. Static IP address assignment configuration example	60
7.6.1.2.2. Dynamic IP address assignment configuration example	60
7.7. CONFIGURING A HOST-DEVICE NETWORK	60
7.7.1. Creating an additional network attachment with the host-device CNI plug-in	60
7.7.1.1. Configuration for host-device	61
7.7.1.1.1. host-device configuration example	63
7.7.1.2. Configuration for ipam CNI plug-in	63
7.7.1.2.1. Static IP address assignment configuration example	64
7.7.1.2.2. Dynamic IP address assignment configuration example	64
7.8. EDITING AN ADDITIONAL NETWORK	65
7.8.1. Modifying an additional network attachment definition	65
7.9. REMOVING AN ADDITIONAL NETWORK	65
7.9.1. Removing an additional network attachment definition	66
7.10. CONFIGURING PTP	66
7.10.1. About PTP hardware on OpenShift Container Platform	67
7.10.2. Installing the PTP Operator	67
7.10.2.1. Installing the Operator using the CLI	67
7.10.2.2. Installing the Operator using the web console	68
7.10.3. Automated discovery of PTP network devices	69
7.10.4. Configuring Linuxptp services	70
CHAPTER 8. HARDWARE NETWORKS	73
8.1. ABOUT SINGLE ROOT I/O VIRTUALIZATION (SR-IOV) HARDWARE NETWORKS	73
8.1.1. About SR-IOV hardware on OpenShift Container Platform	73

8.1.1.1. Supported devices	74
8.1.1.2. Example use of a virtual function (VF) in a Pod	74
8.2. INSTALLING THE SR-IOV NETWORK OPERATOR	75
8.2.1. Installing SR-IOV Network Operator	75
8.2.1.1. Installing the Operator using the CLI	76
8.2.1.2. Installing the Operator using the web console	77
8.3. CONFIGURING THE SR-IOV NETWORK OPERATOR	78
8.3.1. Configuring the SR-IOV Network Operator	78
8.3.1.1. About the Network Resources Injector	79
8.3.1.2. About the SR-IOV Operator admission controller webhook	79
8.3.1.3. About custom node selectors	79
8.3.1.4. Disabling or enabling the Network Resources Injector	79
8.3.1.5. Disabling or enabling the SR-IOV Operator admission controller webhook	80
8.3.1.6. Configuring a custom NodeSelector for the SR-IOV Network Config daemon	80
8.4. CONFIGURING AN SR-IOV NETWORK DEVICE	81
8.4.1. Automated discovery of SR-IOV network devices	81
8.4.2. Configuring SR-IOV network devices	82
8.5. CONFIGURING A SR-IOV NETWORK ATTACHMENT	85
8.5.1. Configuring SR-IOV additional network	85
8.5.1.1. Configuration for ipam CNI plug-in	87
8.5.1.1.1. Static IP address assignment configuration example	88
8.5.1.1.2. Dynamic IP address assignment configuration example	88
8.5.1.2. Configuring static MAC and IP addresses on additional SR-IOV networks	88
8.6. ADDING A POD TO AN SR-IOV ADDITIONAL NETWORK	90
8.6.1. Adding a Pod to an additional network	90
8.6.2. Creating a non-uniform memory access (NUMA) aligned SR-IOV pod	92
8.7. USING HIGH PERFORMANCE MULTICAST	94
8.7.1. Configuring high performance multicast	94
8.7.2. Using an SR-IOV interface for multicast	94
8.8. USING VIRTUAL FUNCTIONS (VFS) WITH DPDK AND RDMA MODES	95
8.8.1. Examples of using virtual functions in DPDK and RDMA modes	95
8.8.1.1. Example use of virtual function (VF) in DPDK mode with Intel NICs	96
8.8.1.2. Example use of a virtual function in DPDK mode with Mellanox NICs	99
8.8.1.3. Example of a virtual function in RDMA mode with Mellanox NICs	102
CHAPTER 9. OPENSIFT SDN NETWORK PROVIDER	105
9.1. ABOUT OPENSIFT SDN	105
9.2. CONFIGURING EGRESS IPS FOR A PROJECT	105
9.2.1. Egress IP address assignment for project egress traffic	105
9.2.1.1. Considerations when using automatically assigned egress IP addresses	106
9.2.1.2. Considerations when using manually assigned egress IP addresses	106
9.2.2. Configuring automatically assigned egress IP addresses for a namespace	106
9.2.3. Configuring manually assigned egress IP addresses for a namespace	108
9.3. CONFIGURING AN EGRESS FIREWALL TO CONTROL ACCESS TO EXTERNAL IP ADDRESSES	109
9.3.1. How an egress firewall works in a project	109
9.3.1.1. Limitations of an egress firewall	110
9.3.1.2. Matching order for egress network policy rules	110
9.3.1.3. How Domain Name Server (DNS) resolution works	110
9.3.2. EgressNetworkPolicy custom resource (CR) object	110
9.3.2.1. EgressNetworkPolicy rules	111
9.3.2.2. Example EgressNetworkPolicy CR object	111
9.3.3. Creating an egress firewall policy object	112
9.4. EDITING AN EGRESS FIREWALL FOR A PROJECT	112

9.4.1. Editing an EgressNetworkPolicy object	112
9.4.2. EgressNetworkPolicy custom resource (CR) object	113
9.4.2.1. EgressNetworkPolicy rules	114
9.4.2.2. Example EgressNetworkPolicy CR object	114
9.5. REMOVING AN EGRESS FIREWALL FROM A PROJECT	114
9.5.1. Removing an EgressNetworkPolicy object	114
9.6. USING MULTICAST	115
9.6.1. About multicast	115
9.6.2. Enabling multicast between Pods	115
9.6.3. Disabling multicast between Pods	116
9.7. CONFIGURING NETWORK ISOLATION USING OPENSIFT SDN	116
9.7.1. Joining projects	117
9.7.2. Isolating a project	117
9.7.3. Disabling network isolation for a project	117
9.8. CONFIGURING KUBE-PROXY	118
9.8.1. About iptables rules synchronization	118
9.8.2. Modifying the kube-proxy configuration	118
9.8.3. kube-proxy configuration parameters	119
CHAPTER 10. CONFIGURING ROUTES	121
10.1. ROUTE CONFIGURATION	121
10.1.1. Configuring route timeouts	121
10.1.2. Enabling HTTP strict transport security	121
10.1.3. Troubleshooting throughput issues	122
10.1.4. Using cookies to keep route statefulness	122
10.1.4.1. Annotating a route with a cookie	123
10.1.5. Route-specific annotations	123
10.2. SECURED ROUTES	125
10.2.1. Creating a re-encrypt route with a custom certificate	125
10.2.2. Creating an edge route with a custom certificate	127
CHAPTER 11. CONFIGURING INGRESS CLUSTER TRAFFIC	129
11.1. CONFIGURING INGRESS CLUSTER TRAFFIC OVERVIEW	129
11.2. CONFIGURING INGRESS CLUSTER TRAFFIC USING AN INGRESS CONTROLLER	129
11.2.1. Using Ingress Controllers and routes	129
11.2.2. Creating a project and service	130
11.2.3. Exposing the service by creating a route	131
11.2.4. Configuring Ingress Controller sharding by using route labels	132
11.2.5. Configuring Ingress Controller sharding by using namespace labels	132
11.2.6. Additional resources	133
11.3. CONFIGURING INGRESS CLUSTER TRAFFIC USING A LOAD BALANCER	133
11.3.1. Using a load balancer to get traffic into the cluster	133
11.3.2. Creating a project and service	134
11.3.3. Exposing the service by creating a route	135
11.3.4. Creating a load balancer service	136
11.4. CONFIGURING INGRESS CLUSTER TRAFFIC USING A SERVICE EXTERNAL IP	137
11.4.1. Using a service external IP to get traffic into the cluster	137
11.4.2. Creating a project and service	138
11.4.3. Exposing the service by creating a route	139
11.5. CONFIGURING INGRESS CLUSTER TRAFFIC USING A NODEPORT	140
11.5.1. Using a NodePort to get traffic into the cluster	140
11.5.2. Creating a project and service	140
11.5.3. Exposing the service by creating a route	141

CHAPTER 12. CONFIGURING THE CLUSTER-WIDE PROXY	143
12.1. ENABLING THE CLUSTER-WIDE PROXY	143
12.2. REMOVING THE CLUSTER-WIDE PROXY	145
CHAPTER 13. CONFIGURING A CUSTOM PKI	147
13.1. CONFIGURING THE CLUSTER-WIDE PROXY DURING INSTALLATION	147
13.2. ENABLING THE CLUSTER-WIDE PROXY	149
13.3. CERTIFICATE INJECTION USING OPERATORS	151

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. OPENSIFT CONTAINER PLATFORM DNS

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

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

CHAPTER 2. ACCESSING HOSTS

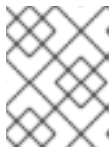
Learn how to create a bastion host to access OpenShift Container Platform instances and access the master nodes with secure shell (SSH) access.

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

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

Procedure

1. Create a security group that allows SSH access into the virtual private cloud (VPC) created by the **openshift-install** command.
2. Create an Amazon EC2 instance on one of the public subnets the installer created.
3. Associate a public IP address with the Amazon EC2 instance that you created.
Unlike with the OpenShift Container Platform installation, you should associate the Amazon EC2 instance you created with an SSH keypair. It does not matter what operating system you choose for this instance, as it will simply serve as an SSH bastion to bridge the internet into your OpenShift Container Platform cluster's VPC. The Amazon Machine Image (AMI) you use does matter. With Red Hat Enterprise Linux CoreOS, for example, you can provide keys via Ignition, like the installer does.
4. Once you provisioned your Amazon EC2 instance and can SSH into it, you must add the SSH key that you associated with your OpenShift Container Platform installation. This key can be different from the key for the bastion instance, but does not have to be.



NOTE

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

5. Run **oc get nodes**, inspect the output, and choose one of the nodes that is a master. The host name looks similar to **ip-10-0-1-163.ec2.internal**.
6. From the bastion SSH host you manually deployed into Amazon EC2, SSH into that master host. Ensure that you use the same SSH key you specified during the installation:

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

CHAPTER 3. CLUSTER NETWORK OPERATOR IN OPENSIFT 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) Pod network provider plug-in selected for the cluster during installation.

3.1. CLUSTER NETWORK OPERATOR

The Cluster Network Operator implements the **network** API from the **operator.openshift.io** API group. The Operator deploys the OpenShift SDN Pod network provider plug-in, or the Pod network provider plug-in that you selected during cluster installation, by using a DaemonSet.

Procedure

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

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

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

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

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

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

NAME	VERSION	AVAILABLE	PROGRESSING	DEGRADED	SINCE
network	4.3.0	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: ❶
  Cluster Network:
    Cidr:      10.128.0.0/14
    Host Prefix: 23
    Network Type: OpenShiftSDN
    Service Network:
      172.30.0.0/16
Status: ❷
  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>

```

- ❶ The **Spec** field displays the configured state of the cluster network.
- ❷ The **Status** field displays the current state of the cluster network configuration.

3.3. VIEWING CLUSTER NETWORK OPERATOR STATUS

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

Procedure

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

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

3.4. VIEWING CLUSTER NETWORK OPERATOR LOGS

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

Procedure

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

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



IMPORTANT

The Open Virtual Networking (OVN) Kubernetes network plug-in is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of the OVN Technology Preview, see <https://access.redhat.com/articles/4380121>.

3.5. CLUSTER NETWORK OPERATOR CONFIGURATION

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

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

Cluster Network Operator CR

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  clusterNetwork: ❶
  - cidr: 10.128.0.0/14
    hostPrefix: 23
  serviceNetwork: ❷
  - 172.30.0.0/16
  defaultNetwork: ❸
  ...
  kubeProxyConfig: ❹
    iptablesSyncPeriod: 30s ❺
    proxyArguments:
      iptables-min-sync-period: ❻
      - 30s
```

- ❶ A list specifying the blocks of IP addresses from which Pod IPs are allocated and the subnet prefix length assigned to each individual node.
- ❷ 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.
- ❸ Configures the Pod network provider for the cluster network.
- ❹ The parameters for this object specify the Kubernetes network proxy (kube-proxy) configuration. If you are using the OVN-Kubernetes network provider, the kube-proxy configuration has no effect.

- 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 the OpenShift SDN network provider

The following YAML object describes the configuration parameters for the OpenShift SDN Pod network provider.



NOTE

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

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

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

3.5.2. Configuration parameters for the OVN-Kubernetes network provider

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



NOTE

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

```
defaultNetwork:
  type: OVNKubernetes 1
  ovnKubernetesConfig: 2
    mtu: 1450 3
    genevePort: 6081 4
```


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

3.5.3. Cluster Network Operator example configuration

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

Cluster Network Operator example CR

```
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 OPENSIFT 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**.

Procedure

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

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

```
Cluster Domain: cluster.local 1
Cluster IP: 172.30.0.10 2
...
```

- 1** The Cluster Domain field is the base DNS domain used to construct fully qualified Pod and Service domain names.
- 2** 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]
```

4.3. USING DNS FORWARDING

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

Procedure

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

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

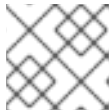
This allows the Operator to create and update the ConfigMap named **dns-default** with additional server configuration blocks based on **Server**. If none of the servers has a zone that matches the query, then name resolution falls back to the name servers that are specified in **/etc/resolv.conf**.

Sample DNS

```
apiVersion: operator.openshift.io/v1
kind: DNS
metadata:
  name: default
spec:
  servers:
    - name: foo-server 1
      zones: 2
        - foo.com
      forwardPlugin:
        upstreams: 3
          - 1.1.1.1
          - 2.2.2.2:5353
    - name: bar-server
      zones:
        - bar.com
        - example.com
      forwardPlugin:
```

```
upstreams:
- 3.3.3.3
- 4.4.4.4:5454
```

- 1 **name** must comply with the **rfc6335** service name syntax.
- 2 **zones** must conform to the definition of a **subdomain** in **rfc1123**. The cluster domain, **cluster.local**, is an invalid **subdomain** for **zones**.
- 3 A maximum of 15 **upstreams** is allowed per **forwardPlugin**.

**NOTE**

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

2. View the ConfigMap:

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

Sample DNS ConfigMap based on previous sample DNS

```
apiVersion: v1
data:
  Corefile: |
    foo.com:5353 {
      forward . 1.1.1.1 2.2.2.2:5353
    }
    bar.com:5353 example.com:5353 {
      forward . 3.3.3.3 4.4.4.4:5454 1
    }
    .:5353 {
      errors
      health
      kubernetes cluster.local in-addr.arpa ip6.arpa {
        pods insecure
        upstream
        fallthrough in-addr.arpa ip6.arpa
      }
      prometheus :9153
      forward . /etc/resolv.conf {
        policy sequential
      }
      cache 30
      reload
    }
kind: ConfigMap
metadata:
  labels:
    dns.operator.openshift.io/owning-dns: default
  name: dns-default
  namespace: openshift-dns
```

- 1 Changes to the **forwardPlugin** triggers a rolling update of the CoreDNS DaemonSet.

Additional resources

- For more information on DNS forwarding, see the [CoreDNS forward documentation](#).

4.4. DNS OPERATOR STATUS

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

Procedure

View the status of the DNS Operator:

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

4.5. DNS OPERATOR LOGS

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

Procedure

View the logs of the DNS Operator:

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

CHAPTER 5. INGRESS OPERATOR IN OPENSIFT 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

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

Parameter	Description
-----------	-------------

Parameter	Description
domain	<p>domain is a DNS name serviced by the Ingress controller and is used to configure multiple features:</p> <ul style="list-style-type: none"> For the LoadBalancerService endpoint publishing strategy, domain is used to configure DNS records. See endpointPublishingStrategy. When using a generated default certificate, the certificate is valid for domain and its subdomains. See defaultCertificate. The value is published to individual Route statuses so that users know where to target external DNS records. <p>The domain value must be unique among all Ingress controllers and cannot be updated.</p> <p>If empty, the default value is ingress.config.openshift.io/cluster.spec.domain.</p>
replicas	<p>replicas is the desired number of Ingress controller replicas. If not set, the default value is 2.</p>
endpointPublishingStrategy	<p>endpointPublishingStrategy is used to publish the Ingress controller endpoints to other networks, enable load balancer integrations, and provide access to other systems.</p> <p>If not set, the default value is based on infrastructure.config.openshift.io/cluster.status.platform:</p> <ul style="list-style-type: none"> AWS: LoadBalancerService (with external scope) Azure: LoadBalancerService (with external scope) GCP: LoadBalancerService (with external scope) Other: HostNetwork. <p>The endpointPublishingStrategy value cannot be updated.</p>
defaultCertificate	<p>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.</p> <p>The secret must contain the following keys and data: * tls.crt: certificate file contents * tls.key: key file contents</p> <p>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.</p> <p>The in-use certificate, whether generated or user-specified, is automatically integrated with OpenShift Container Platform built-in OAuth server.</p>

Parameter	Description
namespaceSelector	namespaceSelector is used to filter the set of namespaces serviced by the Ingress controller. This is useful for implementing shards.
routeSelector	routeSelector is used to filter the set of Routes serviced by the Ingress controller. This is useful for implementing shards.
nodePlacement	<p>nodePlacement enables explicit control over the scheduling of the Ingress controller.</p> <p>If not set, the defaults values are used.</p> <div>  <div> <p>NOTE</p> <p>The nodePlacement parameter includes two parts, nodeSelector and tolerations. For example:</p> <pre>nodePlacement: nodeSelector: matchLabels: beta.kubernetes.io/os: linux tolerations: - effect: NoSchedule operator: Exists</pre> </div> </div>

Parameter	Description
tlsSecurityProfile	<p>tlsSecurityProfile specifies settings for TLS connections for Ingress controllers.</p> <p>If not set, the default value is based on the apiservers.config.openshift.io/cluster resource.</p> <p>When using the Old, Intermediate, and Modern profile types, the effective profile configuration is subject to change between releases. For example, given a specification to use the Intermediate profile deployed on release X.Y.Z, an upgrade to release X.Y.Z+1 may cause a new profile configuration to be applied to the Ingress controller, resulting in a rollout.</p> <p>The minimum TLS version for Ingress controllers is 1.1, and the maximum TLS version is 1.2.</p> <div>  <p>IMPORTANT</p> <p>The HAProxy Ingress controller image does not support TLS 1.3 and because the Modern profile requires TLS 1.3, it is not supported. The Ingress Operator converts the Modern profile to Intermediate.</p> <p>The Ingress Operator also converts the TLS 1.0 of an Old or Custom profile to 1.1, and TLS 1.3 of a Custom profile to 1.2.</p> </div> <div>  <p>NOTE</p> <p>Ciphers and the minimum TLS version of the configured security profile are reflected in the TLSPProfile status.</p> </div>

**NOTE**

All parameters are optional.

5.2.1. Ingress controller TLS profiles

The **tlsSecurityProfile** parameter defines the schema for a TLS security profile. This object is used by operators to apply TLS security settings to operands.

There are four TLS security profile types:

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

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

Sample Old profile configuration

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

Sample Intermediate profile configuration

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

Sample Modern profile configuration

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

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



WARNING

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

Sample Custom profile

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

5.2.2. Ingress controller endpoint publishing strategy

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

Your can view the status of a particular Ingress Controller.

Procedure

- View the status of an Ingress Controller:

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

5.7. SETTING A CUSTOM DEFAULT CERTIFICATE

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

Prerequisites

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

- Your certificate is valid for the Ingress domain.
- You must have an **IngressController** CR. You may use the default one:

```
$ oc --namespace openshift-ingress-operator get ingresscontrollers
NAME    AGE
default 10m
```



NOTE

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

Procedure

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



NOTE

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

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

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

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

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

3. Verify the update was effective:

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

The output should look like:

```
map[name:custom-certs-default]
```

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

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

5.8. SCALING AN INGRESS CONTROLLER

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

Procedure

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

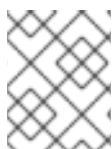
```
$ oc get -n openshift-ingress-operator ingresscontrollers/default -o
jsonpath='{$.status.availableReplicas}'
2
```

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

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

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

```
$ oc get -n openshift-ingress-operator ingresscontrollers/default -o
jsonpath='{$.status.availableReplicas}'
3
```



NOTE

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

5.9. CONFIGURING INGRESS CONTROLLER SHARDING BY USING ROUTE LABELS

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

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

Procedure

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

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

```

domain: <apps-sharded.basedomain.example.net>
nodePlacement:
  nodeSelector:
    matchLabels:
      node-role.kubernetes.io/worker: ""
routeSelector:
  matchLabels:
    type: sharded
status: {}
kind: List
metadata:
  resourceVersion: ""
  selfLink: ""

```

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

```
# 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:

```

# 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**.

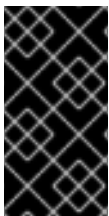
5.11. CONFIGURING AN INGRESS CONTROLLER TO USE AN INTERNAL LOAD BALANCER

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



WARNING

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



IMPORTANT

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

See the [Kubernetes Services documentation](#) for implementation details.

Prerequisites

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

Procedure

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

```

apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  namespace: openshift-ingress-operator

```

```
name: <name> ❶
spec:
  domain: <domain> ❷
  endpointPublishingStrategy:
    type: LoadBalancerService
  loadBalancer:
    scope: Internal ❸
```

- ❶ Replace **<name>** with a name for the **IngressController** object.
- ❷ Specify the **domain** for the application published by the controller.
- ❸ Specify a value of **Internal** to use an internal load balancer.

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

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

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

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

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

5.12. CONFIGURING THE DEFAULT INGRESS CONTROLLER FOR YOUR CLUSTER TO BE INTERNAL

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



WARNING

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



IMPORTANT

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

Prerequisites

- Install the OpenShift command-line interface (CLI), commonly known as **oc**.

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

Procedure

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

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

5.13. ADDITIONAL RESOURCES

- [Configuring a custom PKI](#)

CHAPTER 6. CONFIGURING NETWORK POLICY WITH OPENSIFT SDN

6.1. ABOUT NETWORK POLICY

In a cluster using a Kubernetes Container Network Interface (CNI) plug-in that supports Kubernetes network policy, network isolation is controlled entirely by NetworkPolicy Custom Resource (CR) objects. In OpenShift Container Platform 4.3, OpenShift SDN supports using NetworkPolicy in its default network isolation mode.



NOTE

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



WARNING

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

By default, all Pods in a project are accessible from other Pods and network endpoints. To isolate one or more Pods in a project, you can create NetworkPolicy objects in that project to indicate the allowed incoming connections. Project administrators can create and delete NetworkPolicy objects within their own project.

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

The following example NetworkPolicy objects demonstrate supporting different scenarios:

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

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

- Only allow connections from the OpenShift Container Platform Ingress Controller:
To make a project allow only connections from the OpenShift Container Platform Ingress Controller, add the following NetworkPolicy object:

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

```

kind: NetworkPolicy
metadata:
  name: allow-from-openshift-ingress
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          network.openshift.io/policy-group: ingress
  podSelector: {}
  policyTypes:
  - Ingress

```

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

- Only accept connections from Pods within a project:
To make Pods accept connections from other Pods in the same project, but reject all other connections from Pods in other projects, add the following NetworkPolicy object:

```

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

```

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

```

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

```

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

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

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

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

6.2. EXAMPLE NETWORKPOLICY OBJECT

The following annotates an example NetworkPolicy object:

```
kind: NetworkPolicy
apiVersion: extensions/v1beta1
metadata:
  name: allow-27107 1
spec:
  podSelector: 2
    matchLabels:
      app: mongodb
  ingress:
    - from:
      - podSelector: 3
          matchLabels:
            app: app
      ports: 4
        - protocol: TCP
          port: 27017
```

1 The **name** of the NetworkPolicy object.

2

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

- 3 A selector matching the Pods that the policy object allows ingress traffic from. The selector will match Pods in any project.
- 4 A list of one or more destination ports to accept traffic on.

6.3. CREATING A NETWORKPOLICY OBJECT

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

Prerequisites

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

Procedure

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

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: <policy-name> 1
spec:
  podSelector:
  ingress: []
```

- 1 Specify a name for the policy object.

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

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

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

```
$ 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:

```
$ oc delete networkpolicy -l name=<policy-name> 1
```

- 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:

```
$ 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:

```
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:

```
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 \ ❶
-n <project> ❷
```

- ❶ Replace **<policy-name>** with the filename of the file containing the policy.
- ❷ Replace **<project>** with the name of the project to apply the NetworkPolicy object to.

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

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

```
$ oc get --namespace openshift-ingress-operator ingresscontrollers/default \
--output jsonpath='{.status.endpointPublishingStrategy.type}'
```

- b. If the previous command reports the endpoint publishing strategy as **HostNetwork**, set a label on the **default** namespace:

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

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

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

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

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

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

6.7. CREATING DEFAULT NETWORK POLICIES FOR A NEW PROJECT

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

6.7.1. Modifying the template for new projects

As a cluster administrator, you can modify the default project template so that new projects are created using your custom requirements.

To create your own custom project template:

Procedure

1. Log in as a user with **cluster-admin** privileges.
2. Generate the default project template:

```
$ oc adm create-bootstrap-project-template -o yaml > template.yaml
```

3. Use a text editor to modify the generated **template.yaml** file by adding objects or modifying existing objects.

4. The project template must be created in the **openshift-config** namespace. Load your modified template:

```
$ oc create -f template.yaml -n openshift-config
```

5. Edit the project configuration resource using the web console or CLI.

- Using the web console:
 - i. Navigate to the **Administration** → **Cluster Settings** page.
 - ii. Click **Global Configuration** to view all configuration resources.
 - iii. Find the entry for **Project** and click **Edit YAML**.
- Using the CLI:
 - i. Edit the **project.config.openshift.io/cluster** resource:

```
$ oc edit project.config.openshift.io/cluster
```

6. Update the **spec** section to include the **projectRequestTemplate** and **name** parameters, and set the name of your uploaded project template. The default name is **project-request**.

Project configuration resource with custom project template

```
apiVersion: config.openshift.io/v1
kind: Project
metadata:
  ...
spec:
  projectRequestTemplate:
    name: <template_name>
```

7. After you save your changes, create a new project to verify that your changes were successfully applied.

6.7.2. Adding network policy objects to the new project template

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

Prerequisites

- A cluster using the OpenShift SDN network plug-in with **mode: NetworkPolicy** set. This mode is the default for OpenShift SDN.
- Install the OpenShift Command-line Interface (CLI), commonly known as **oc**.
- You must log in to the cluster with a user with **cluster-admin** privileges.
- You must have created a custom default project template for new projects.

Procedure

1. Edit the default template for a new project by running the following command:

```
$ oc edit template <project_template> -n openshift-config
```

Replace **<project_template>** with the name of the default template that you configured for your cluster. The default template name is **project-request**.

2. In the template, add each NetworkPolicy object as an element to the **objects** parameter. The **objects** parameter accepts a collection of one or more objects. In the following example, the **objects** parameter collection includes several NetworkPolicy objects:

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

3. Optional: Create a new project to confirm that your network policy objects are created successfully by running the following commands:

- a. Create a new project:

```
$ oc new-project <project> 1
```

1 Replace **<project>** with the name for the project you are creating.

- b. Confirm that the network policy objects in the new project template exist in the new project:

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

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

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 with additional networks, complete the following steps:

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

```
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: <network>[,<network>,...] 1
```

- 1 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:

```
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 20000000000000"]
    image: centos/tools
```

2. Create the Pod by running the following command:

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

3. Optional: Confirm that the annotation exists in the Pod CR by running the following command. Replace **<name>** with the name of the Pod.

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

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

```
$ oc get pod example-pod -o yaml
apiVersion: v1
kind: Pod
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: macvlan-bridge
    k8s.v1.cni.cncf.io/networks-status: |- ❶
    [{
      "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:
  ...
```

- ❶ 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.

7.2.1.1. Specifying Pod-specific addressing and routing options

When attaching a Pod to an additional network, you may want to specify further properties about that network in a particular Pod. This allows you to change some aspects of routing, as well as specify static IP addresses and MAC addresses. In order to accomplish this, you can use the JSON formatted annotations.

Prerequisites

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

Procedure

To add a Pod to an additional network while specifying addressing and/or routing options, complete the following steps:

1. Edit the Pod resource definition. If you are editing an existing Pod, run the following command to edit its definition in the default editor. Replace **<name>** with the name of the Pod to edit.

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

2. In the Pod resource definition, add the **k8s.v1.cni.cncf.io/networks** parameter to the Pod **metadata** mapping. The **k8s.v1.cni.cncf.io/networks** accepts a JSON string of a list of objects that reference the name of NetworkAttachmentDefinition Custom Resource (CR) names in addition to specifying additional properties.

```
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: ' [<network>,<network>,...]' 1
```

- 1 Replace **<network>** with a JSON object as shown in the following examples. The single quotes are required.

3. In the following example the annotation specifies which network attachment will have the default route, using the **default-route** parameter.

```
apiVersion: v1
kind: Pod
metadata:
  name: example-pod
  annotations:
    k8s.v1.cni.cncf.io/networks: '
    {
      "name": "net1"
    },
    {
      "name": "net2", 1
      "default-route": ["192.0.2.1"] 2
    }
  '
spec:
  containers:
  - name: example-pod
    command: ["/bin/bash", "-c", "sleep 20000000000000"]
    image: centos/tools
```

- 1 The **name** key is the name of the additional network to associate with the Pod.
- 2 The **default-route** key specifies a value of a gateway for traffic to be routed over if no

other routing entry is present in the routing table. If more than one **default-route** key is specified, this will cause the Pod to fail to become active.

The default route will cause any traffic that is not specified in other routes to be routed to the gateway.



IMPORTANT

Setting the default route to an interface other than the default network interface for OpenShift Container Platform may cause traffic that is anticipated for Pod-to-Pod traffic to be routed over another interface.

To verify the routing properties of a Pod, the **oc** command may be used to execute the **ip** command within a Pod.

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



NOTE

You may also reference the Pod's **k8s.v1.cni.cncf.io/networks-status** to see which additional network has been assigned the default route, by the presence of the **default-route** key in the JSON-formatted list of objects.

To set a static IP address or MAC address for a Pod you can use the JSON formatted annotations. This requires you create networks that specifically allow for this functionality. This can be specified in a rawCNConfig for the CNO.

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

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

The following YAML describes the configuration parameters for the CNO:

Cluster Network Operator YAML configuration

```
name: <name> ①
namespace: <namespace> ②
rawCNConfig: '{ ③
  ...
}'
type: Raw
```

- ① Specify a name for the additional network attachment that you are creating. The name must be unique within the specified **namespace**.
- ② Specify the namespace to create the network attachment in. If you do not specify a value, then the **default** namespace is used.
- ③ Specify the CNI plug-in configuration in JSON format, which is based on the following template.

The following object describes the configuration parameters for utilizing static MAC address and IP address using the macvlan CNI plug-in:

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

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

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

The above network attachment may then be referenced in a JSON formatted annotation, along with keys to specify which static IP and MAC address will be assigned to a given Pod.

Edit the desired Pod with:

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

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

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

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

- 3 Provide the desired MAC address.



NOTE

Static IP addresses and MAC addresses do not have to be used at the same time, you may use them individually, or together.

To verify the IP address and MAC properties of a Pod with additional networks, use the **oc** command to execute the `ip` command within a Pod.

```
$ oc exec -it <pod_name> -- ip a
```

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.

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

2. Update the **annotations** mapping to remove the additional network from the Pod by performing one of the following actions:
 - To remove all additional networks from a Pod, remove the **k8s.v1.cni.cncf.io/networks** parameter from the Pod resource definition as in the following example:

```
apiVersion: v1
kind: Pod
metadata:
  name: example-pod
  annotations: {}
spec:
  containers:
```

```
- 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.

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

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

```
$ oc describe pod example-pod

Name:          example-pod
...
Annotations:   k8s.v1.cni.cncf.io/networks-status:
                [{
                  "name": "openshift-sdn",
                  "interface": "eth0",
                  "ips": [
                    "10.131.0.13"
                  ],
                  "default": true, 1
                  "dns": {}
                }]
Status:        Running
...
```

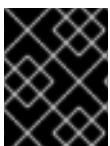
1 Only the default cluster network is attached to the Pod.

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:

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

- ❶ Specify the configuration for the additional network attachment definition.

3. Save your changes and quit the text editor to commit your changes.
4. Optional: Confirm that the CNO created the NetworkAttachmentDefinition CR by running the following command. There might be a delay before the CNO creates the CR.

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

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

```
name: <name> ❶
namespace: <namespace> ❷
rawCNIConfig: '{ ❸
  ...
}'
type: Raw
```

- ❶ Specify a name for the additional network attachment that you are creating. The name must be unique within the specified **namespace**.
- ❷ Specify the namespace to create the network attachment in. If you do not specify a value, then the **default** namespace is used.
- ❸ 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

```
{
  "cniVersion": "0.3.1",
  "name": "<name>", ❶
  "type": "bridge",
  "bridge": "<bridge>", ❷
  "ipam": { ❸
    ...
  },
  "ipMasq": false, ❹
  "isGateway": false, ❺
  "isDefaultGateway": false, ❻
  "forceAddress": false, ❼
  "hairpinMode": false, ❽
  "promiscMode": false, ❾
  "vlan": <vlan>, ❿
  "mtu": <mtu> ❶❶
}
```

- ❶ Specify the value for the **name** parameter you provided previously for the CNO configuration.
- ❷ 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**.
- 9 Set to **true** to enable promiscuous mode on the bridge. The default value is **false**.
- 10 Specify a virtual LAN (VLAN) tag as an integer value. By default, no VLAN tag is assigned.
- 11 Set the maximum transmission unit (MTU) to the specified value. The default value is automatically set by the kernel.

7.4.1.1.1. bridge configuration example

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

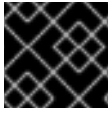
```
name: bridge-net
namespace: work-network
type: Raw
rawCNICfg: '{
  "cniVersion": "0.3.1",
  "type": "bridge",
  "master": "eth1",
  "isGateway": true,
  "vlan": 2,
  "ipam": {
    "type": "dhcp"
  }
}'
```

- 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.

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>", 1
    "addresses": [ 2
      {
        "address": "<address>", 3
        "gateway": "<gateway>" 4
      }
    ],
    "routes": [ 5
      {
        "dst": "<dst>" 6
        "gw": "<gw>" 7
      }
    ],
    "dns": { 8
      "nameservers": ["<nameserver>"], 9
      "domain": "<domain>", 10
      "search": ["<search_domain>"] 11
    }
  }
}
```

- 1 Specify **static** to configure the plug-in to manage IP address assignment. Specify **DHCP** to allow a DHCP server to manage IP address assignment. You cannot specify any additional parameters if you specify a value of **DHCP**.
- 2 An array describing IP addresses to assign to the virtual interface. Both IPv4 and IPv6 IP addresses are supported.
- 3 A block of IP addresses that you specify in CIDR format to assign to Pods on a worker node, such as **10.1.1.0/24**.
- 4 The default gateway to route egress network traffic to.
- 5 An array describing routes to configure inside the Pod.
- 6 The IP address range in CIDR format.
- 7 The gateway 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.

7.4.1.2.1. Static IP address assignment configuration example

You can configure ipam for static IP address assignment:

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

7.4.1.2.2. Dynamic IP address assignment configuration example

You can configure ipam for DHCP:

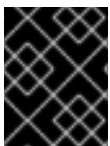
```
{
  "ipam": {
    "type": "DHCP"
  }
}
```

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:

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

- 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.

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

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> ❶
namespace: <namespace> ❷
type: SimpleMacvlan
simpleMacvlanConfig:
  master: <master> ❸
  mode: <mode> ❹
  mtu: <mtu> ❺
  ipamConfig: ❻
  ...
```

- ❶ Specify a name for the additional network attachment that you are creating. The name must be unique within the specified **namespace**.
- ❷ Specify the namespace to create the network attachment in. If a value is not specified, the **default** namespace is used.
- ❸ 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.
- ❹ 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**.
- ❺ Set the maximum transmission unit (MTU) to the specified value. The default value is automatically set by the kernel.
- ❻ 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

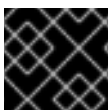
The following example configures an additional network named **macvlan-net**:

```
name: macvlan-net
namespace: work-network
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.

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

```
ipamConfig:
  type: <type> 1
  ... 2
```

- 1 Specify **static** to configure the plug-in to manage IP address assignment. Specify **DHCP** to allow a DHCP server to manage IP address assignment. You cannot specify any additional parameters if you specify a value of **DHCP**.
- 2 If you set the **type** parameter to **static**, then provide the **staticIPAMConfig** parameter.

7.5.1.2.1. Static ipam configuration YAML

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

Static ipam configuration YAML

```
ipamConfig:
  type: static
  staticIPAMConfig:
    addresses: 1
    - address: <address> 2
      gateway: <gateway> 3
    routes: 4
    - destination: <destination> 5
      gateway: <gateway> 6
    dns: 7
    nameservers: 8
    - <nameserver>
    domain: <domain> 9
    search: 10
    - <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

```
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:

```
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:

```
ipamConfig:
  type: DHCP
```

Next steps

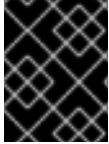
- [Attach a Pod to an additional network](#) .

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:

```
$ 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:

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  additionalNetworks: 1
  - name: test-network-1
    namespace: test-1
    type: Raw
    rawCNICfg: '{
      "cniVersion": "0.3.1",
      "type": "ipvlan",
      "master": "eth1",
      "mode": "l2",
      "ipam": {
        "type": "static",
        "addresses": [
          {
            "address": "191.168.1.1/24"
          }
        ]
      }
    }'
```

```

    ]
  }
}'

```

- 1 Specify the configuration for the additional network attachment definition.
3. Save your changes and quit the text editor to commit your changes.
4. Optional: Confirm that the CNO created the NetworkAttachmentDefinition CR by running the following command. There might be a delay before the CNO creates the CR.

```

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

```

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

```

name: <name> 1
namespace: <namespace> 2
rawCNIConfig: '{ 3
  ...
}'
type: Raw

```

- 1 Specify a name for the additional network attachment that you are creating. The name must be unique within the specified **namespace**.
- 2 Specify the namespace to create the network attachment in. If you do not specify a value, then the **default** namespace is used.
- 3 Specify the CNI plug-in configuration in JSON format, which is based on the following template.

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

ipvlan CNI plug-in JSON configuration object

```

{
  "cniVersion": "0.3.1",

```

```

"name": "<name>", ❶
"type": "ipvlan",
"mode": "<mode>", ❷
"master": "<master>", ❸
"mtu": <mtu>, ❹
"ipam": { ❺
  ...
}
}

```

- ❶ Specify the value for the **name** parameter you provided previously for the CNO configuration.
- ❷ Specify the operating mode for the virtual network. The value must be **I2**, **I3**, or **I3s**. The default value is **I2**.
- ❸ 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.
- ❹ Set the maximum transmission unit (MTU) to the specified value. The default value is automatically set by the kernel.
- ❺ 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**:

```

name: ipvlan-net
namespace: work-network
type: Raw
rawCNIConfig: '{ ❶
  "cniVersion": "0.3.1",
  "type": "ipvlan",
  "master": "eth1",
  "mode": "I3",
  "ipam": {
    "type": "dhcp"
  }
}'

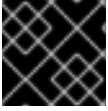
```

- ❶ 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.

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>", 1
    "addresses": [ 2
      {
        "address": "<address>", 3
        "gateway": "<gateway>" 4
      }
    ],
    "routes": [ 5
      {
        "dst": "<dst>" 6
        "gw": "<gw>" 7
      }
    ],
    "dns": { 8
      "nameservers": ["<nameserver>"], 9
      "domain": "<domain>", 10
      "search": ["<search_domain>"] 11
    }
  }
}
```

- 1 Specify **static** to configure the plug-in to manage IP address assignment. Specify **DHCP** to allow a DHCP server to manage IP address assignment. You cannot specify any additional parameters if you specify a value of **DHCP**.
- 2 An array describing IP addresses to assign to the virtual interface. Both IPv4 and IPv6 IP addresses are supported.
- 3 A block of IP addresses that you specify in CIDR format to assign to Pods on a worker node, such as **10.1.1.0/24**.
- 4 The default gateway to route egress network traffic to.
- 5 An array describing routes to configure inside the Pod.
- 6 The IP address range in CIDR format.
- 7 The gateway 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:

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

7.6.1.2.2. Dynamic IP address assignment configuration example

You can configure ipam for DHCP:

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

Next steps

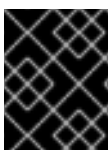
- [Attach a Pod to an additional network](#) .

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:

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

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

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

```
name: <name> ❶
namespace: <namespace> ❷
rawCNICfg: '{ ❸
  ...
}'
type: Raw
```

- ❶ Specify a name for the additional network attachment that you are creating. The name must be unique within the specified **namespace**.
- ❷ Specify the namespace to create the network attachment in. If you do not specify a value, then the **default** namespace is used.
- ❸ 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

```
{
  "cniVersion": "0.3.1",
  "name": "<name>", ❶
  "type": "host-device",
  "device": "<device>", ❷
  "hwaddr": "<hwaddr>", ❸
  "kernelpath": "<kernelpath>", ❹
  "pciBusID": "<pciBusID>", ❺
  "ipam": { ❻
    ...
  }
}
```

- ❶ Specify the value for the **name** parameter you provided previously for the CNO configuration.
- ❷ Specify the name of the device, such as **eth0**.
- ❸ Specify the device hardware MAC address.
- ❹ Specify the Linux kernel device path, such as **/sys/devices/pci0000:00/0000:00:1f.6**.
- ❺ Specify the PCI address of the network device, such as **0000:00:1f.6**.
- ❻ Specify a configuration object for the ipam CNI plug-in. The plug-in manages IP address assignment for the attachment definition.

7.7.1.1.1. host-device configuration example

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

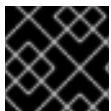
```
name: hostdev-net
namespace: work-network
type: Raw
rawCNConfig: '{
  "cniVersion": "0.3.1",
  "type": "host-device",
  "device": "eth1"
}'
```

1 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.

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>", 1
    "addresses": [ 2
      {
        "address": "<address>", 3
        "gateway": "<gateway>" 4
      }
    ],
    "routes": [ 5
      {
        "dst": "<dst>" 6
        "gw": "<gw>" 7
      }
    ],
    "dns": { 8
      "nameservers": ["<nameserver>"], 9
      "domain": "<domain>", 10
      "search": ["<search_domain>"] 11
    }
  }
}
```

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

```
{
  "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:

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

Next steps

- [Attach a Pod to an additional network](#) .

7.8. EDITING AN ADDITIONAL NETWORK

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

7.8.1. Modifying an additional network attachment definition

As a cluster administrator, you can make changes to an existing additional network. Any existing Pods attached to the additional network will not be updated.

Prerequisites

- You have configured an additional network for your cluster.
- Install the OpenShift Command-line Interface (CLI), commonly known as **oc**.
- Log in as a user with **cluster-admin** privileges.

Procedure

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

1. Run the following command to edit the Cluster Network Operator (CNO) CR in your default text editor:

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

2. In the **additionalNetworks** collection, update the additional network with your changes.
3. Save your changes and quit the text editor to commit your changes.
4. Optional: Confirm that the CNO updated the NetworkAttachmentDefinition CR by running the following command. Replace **<network-name>** with the name of the additional network to display. There might be a delay before the CNO updates the NetworkAttachmentDefinition CR to reflect your changes.

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

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

```
$ oc get network-attachment-definitions net1 -o go-template='{{printf "%s\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"]}}} }
```

7.9. REMOVING AN ADDITIONAL NETWORK

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

7.9.1. Removing an additional network attachment definition

As a cluster administrator, you can remove an additional network from your OpenShift Container Platform cluster. The additional network is not removed from any Pods it is attached to.

Prerequisites

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

Procedure

To remove an additional network from your cluster, complete the following steps:

1. Edit the Cluster Network Operator (CNO) in your default text editor by running the following command:

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

2. Modify the CR by removing the configuration from the **additionalNetworks** collection for the network attachment definition you are removing.

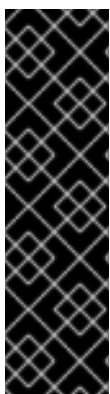
```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  additionalNetworks: [] 1
```

- 1 If you are removing the configuration mapping for the only additional network attachment definition in the **additionalNetworks** collection, you must specify an empty collection.

3. Save your changes and quit the text editor to commit your changes.
4. Optional: Confirm that the additional network CR was deleted by running the following command:

```
$ oc get network-attachment-definition --all-namespaces
```

7.10. CONFIGURING PTP



IMPORTANT

Precision Time Protocol (PTP) hardware is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see <https://access.redhat.com/support/offerings/techpreview/>.

7.10.1. About PTP hardware on OpenShift Container Platform

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

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

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

7.10.2. Installing the PTP Operator

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

7.10.2.1. Installing the Operator using the CLI

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

Prerequisites

- A cluster installed on bare-metal hardware with nodes that have hardware that supports PTP.
- The OpenShift Container Platform Command-line Interface (CLI), commonly known as **oc**.
- Log in as a user with **cluster-admin** privileges.

Procedure

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

```
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-ptp
labels:
  openshift.io/run-level: "1"
```

- b. Create the namespace by running the following command:

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

2. Install the PTP Operator in the namespace you created in the previous step by creating the following objects:
 - a. Create the following OperatorGroup CR and save the YAML in the **ptp-operatorgroup.yaml** file:

```
apiVersion: operators.coreos.com/v1
```

```
kind: OperatorGroup
metadata:
  name: ptp-operators
  namespace: openshift-ptp
spec:
  targetNamespaces:
    - openshift-ptp
```

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

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

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

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

4.3

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

Example Subscription

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

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

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

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

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

- f. Change to the **openshift-ptp** project:

```
$ oc project openshift-ptp
```

Now using project "openshift-ptp"

7.10.2.2. Installing the Operator using the web console

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

**NOTE**

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

Procedure

1. Install the PTP Operator using the OpenShift Container Platform web console:
 - a. In the OpenShift Container Platform web console, click **Operators → OperatorHub**.
 - b. Choose **PTP Operator** from the list of available Operators, and then click **Install**.
 - c. On the **Create Operator Subscription** page, under **A specific namespace on the cluster** select **openshift-ptp**. Then, click **Subscribe**.
2. Optional: Verify that the PTP Operator installed successfully:
 - a. Switch to the **Operators → Installed Operators** page.
 - b. Ensure that **PTP Operator** is listed in the **openshift-ptp** project with a **Status** of **InstallSucceeded**.

**NOTE**

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

If the operator does not appear as installed, to troubleshoot further:

- Go to the **Operators → Installed Operators** page and inspect the **Operator Subscriptions** and **Install Plans** tabs for any failure or errors under **Status**.
- Go to the **Workloads → Pods** page and check the logs for Pods in the **openshift-ptp** project.

7.10.3. Automated discovery of PTP network devices

The PTP Operator adds the **NodePtpDevice.ptp.openshift.io** Custom Resource Definition (CRD) to OpenShift Container Platform. The PTP Operator will search your cluster for PTP capable network devices on each node. The Operator creates and updates a **NodePtpDevice** Custom Resource (CR) for each node that provides a compatible PTP device.

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

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

```
apiVersion: ptp.openshift.io/v1
kind: NodePtpDevice
metadata:
  creationTimestamp: "2019-11-15T08:57:11Z"
  generation: 1
  name: dev-worker-0 1
```

```

namespace: openshift-ptp 2
resourceVersion: "487462"
selfLink: /apis/ptp.openshift.io/v1/namespaces/openshift-ptp/nodeptpdevices/dev-worker-0
uid: 08d133f7-aae2-403f-84ad-1fe624e5ab3f
spec: {}
status:
  devices: 3
    - name: eno1
    - name: eno2
    - name: ens787f0
    - name: ens787f1
    - name: ens801f0
    - name: ens801f1
    - name: ens802f0
    - name: ens802f1
    - name: ens803

```

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

7.10.4. Configuring Linuxptp services

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

Prerequisites

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

Procedure

1. Create the following **PtpConfig** CR, and then save the YAML in the **<name>-ptp-config.yaml** file. Replace **<name>** with the name for this configuration.

```

apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: <name> 1
  namespace: openshift-ptp 2
spec:
  profile: 3
    - name: "profile1" 4
      interface: "ens787f1" 5
      ptp4lOpts: "-s -2" 6

```

```

    phc2sysOpts: "-a -r" 7
    recommend: 8
    - profile: "profile1" 9
      priority: 10 10
      match: 11
      - nodeLabel: "node-role.kubernetes.io/worker" 12
        nodeName: "dev-worker-0" 13

```

- 1 Specify a name for the **PtpConfig** CR.
- 2 Specify the namespace where the PTP Operator is installed.
- 3 Specify an array of one or more **profile** objects.
- 4 Specify the name of a profile object which is used to uniquely identify a profile object.
- 5 Specify the network interface name to use by the **ptp4l** service, for example **ens787f1**.
- 6 Specify system config options for the **ptp4l** service, for example **-s -2**. This should not include the interface name **-i <interface>** and service config file **-f /etc/ptp4l.conf** because these will be automatically appended.
- 7 Specify system config options for the **phc2sys** service, for example **-a -r**.
- 8 Specify an array of one or more **recommend** objects which define rules on how the **profile** should be applied to nodes.
- 9 Specify the **profile** object name defined in the **profile** section.
- 10 Specify the **priority** with an integer value between **0** and **99**. A larger number gets lower priority, so a priority of **99** is lower than a priority of **10**. If a node can be matched with multiple profiles according to rules defined in the **match** field, the profile with the higher priority will be applied to that node.
- 11 Specify **match** rules with **nodeLabel** or **nodeName**.
- 12 Specify **nodeLabel** with the **key** of **node.Labels** from the node object.
- 13 Specify **nodeName** with **node.Name** from the node object.

2. Create the CR by running the following command:

```
$ oc create -f <filename> 1
```

- 1 Replace **<filename>** with the name of the file you created in the previous step.

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

```

$ oc get pods -n openshift-ptp -o wide
NAME                                READY STATUS  RESTARTS  AGE  IP              NODE
NOMINATED NODE  READINESS GATES
linuxptp-daemon-4xkbb             1/1   Running    0       43m  192.168.111.15  dev-worker-0
<none>                           <none>

```

```

linuxptp-daemon-tdspf      1/1   Running 0      43m 192.168.111.11 dev-master-0
<none>                    <none>
ptp-operator-657bbb64c8-2f8sj 1/1   Running 0      43m 10.128.0.116 dev-master-0
<none>                    <none>

```

```

$ oc logs linuxptp-daemon-4xkbb -n openshift-ptp
l1115 09:41:17.117596 4143292 daemon.go:107] in applyNodePTPProfile
l1115 09:41:17.117604 4143292 daemon.go:109] updating NodePTPProfile to:
l1115 09:41:17.117607 4143292 daemon.go:110] -----
l1115 09:41:17.117612 4143292 daemon.go:102] Profile Name: profile1 1
l1115 09:41:17.117616 4143292 daemon.go:102] Interface: ens787f1 2
l1115 09:41:17.117620 4143292 daemon.go:102] Ptp4lOpts: -s -2 3
l1115 09:41:17.117623 4143292 daemon.go:102] Phc2sysOpts: -a -r 4
l1115 09:41:17.117626 4143292 daemon.go:116] -----
l1115 09:41:18.117934 4143292 daemon.go:186] Starting phc2sys...
l1115 09:41:18.117985 4143292 daemon.go:187] phc2sys cmd: &{Path:/usr/sbin/phc2sys
Args:[/usr/sbin/phc2sys -a -r] Env:[] Dir: Stdin:<nil> Stdout:<nil> Stderr:<nil> ExtraFiles:[]
SysProcAttr:<nil> Process:<nil> ProcessState:<nil> ctx:<nil> lookPathErr:<nil> finished:false
childFiles:[] closeAfterStart:[] closeAfterWait:[] goroutine:[] errch:<nil> waitDone:<nil>}
l1115 09:41:19.118175 4143292 daemon.go:186] Starting ptp4l...
l1115 09:41:19.118209 4143292 daemon.go:187] ptp4l cmd: &{Path:/usr/sbin/ptp4l Args:
[/usr/sbin/ptp4l -m -f /etc/ptp4l.conf -i ens787f1 -s -2] Env:[] Dir: Stdin:<nil> Stdout:<nil>
Stderr:<nil> ExtraFiles:[] SysProcAttr:<nil> Process:<nil> ProcessState:<nil> ctx:<nil>
lookPathErr:<nil> finished:false childFiles:[] closeAfterStart:[] closeAfterWait:[] goroutine:[]
errch:<nil> waitDone:<nil>}
ptp4l[102189.864]: selected /dev/ptp5 as PTP clock
ptp4l[102189.886]: port 1: INITIALIZING to LISTENING on INIT_COMPLETE
ptp4l[102189.886]: port 0: INITIALIZING to LISTENING on INIT_COMPLETE

```

- 1** **Profile Name** is the name that is applied to node **dev-worker-0**.
- 2** **Interface** is the PTP device specified in the **profile1** interface field. The **ptp4l** service runs on this interface.
- 3** **Ptp4lOpts** are the ptp4l sysconfig options specified in **profile1** Ptp4lOpts field.
- 4** **Phc2sysOpts** are the phc2sys sysconfig options specified in **profile1** Phc2sysOpts field.

CHAPTER 8. HARDWARE NETWORKS

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

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

8.1.1. About SR-IOV hardware on OpenShift Container Platform

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

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

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

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

- The SR-IOV Operator is a Kubernetes Deployment that manages all SR-IOV components in a cluster. It watches creation, update and deletion of Operator Custom Resources and takes corresponding actions such as generating NetworkAttachmentDefinition Custom Resources for SR-IOV CNI, creating and updating configuration of SR-IOV network device plug-in, creating node specific SrioNetworkNodeState Custom Resources and updating Spec.Interfaces field in each SrioNetworkNodeState Custom Resource, etc.
- The SR-IOV network config daemon is a Kubernetes DaemonSet deployed on worker nodes when SR-IOV Operator is launched. It is responsible for discovering and initializing SR-IOV network devices in cluster.
- The Operator webhook is a Kubernetes Dynamic Admission Controller Webhook that validates correctness of Operator Custom Resource and sets default values for fields in Operator Custom Resource that are not configured by user.
- The Network resources injector is a Kubernetes Dynamic Admission Controller Webhook that provides functionality for patching Kubernetes Pod specifications with requests and limits for custom network resources such as SR-IOV VFs.
- The SR-IOV network device plug-in is a Kubernetes device plug-in for discovering, advertising, and allocating SR-IOV network virtual function (VF) resources. Device plug-ins are used in Kubernetes to enable the use of limited resources, typically in physical devices. Device plug-ins give the Kubernetes scheduler awareness of resource availability, so the scheduler can schedule Pods on nodes with sufficient resources.

- The SR-IOV CNI plug-in plumbs VF interfaces allocated from the SR-IOV device plug-in directly into a Pod.



NOTE

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

8.1.1.1. 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**

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

You can run a remote direct memory access (RDMA) or a Data Plane Development Kit (DPDK) application in a Pod with SR-IOV VF attached.

This example shows a Pod using a VF in RDMA mode:

```
apiVersion: v1
kind: Pod
metadata:
  name: rdma-app
  annotations:
    k8s.v1.cni.cncf.io/networks: sriov-rdma-mlnx
spec:
  containers:
  - name: testpmd
    image: <RDMA_image>
    imagePullPolicy: IfNotPresent
    securityContext:
      capabilities:
        add: ["IPC_LOCK"]
    command: ["sleep", "infinity"]
```

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

```
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

```

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

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

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

Next steps

- [Installing the SR-IOV Network Operator](#)
- Optional: [Configuring the SR-IOV Network Operator](#)
- [Configuring an SR-IOV network device](#)
- [Configuring an SR-IOV network attachment](#)
- [Adding a Pod to an SR-IOV additional network](#)

8.2. INSTALLING THE SR-IOV NETWORK OPERATOR

You can install the Single Root I/O Virtualization (SR-IOV) Network Operator on your cluster to manage SR-IOV network devices and network attachments.

8.2.1. Installing SR-IOV Network Operator

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

8.2.1.1. Installing the Operator using the CLI

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

Prerequisites

- A cluster installed on bare-metal hardware with nodes that have hardware that supports SR-IOV.
- The OpenShift Container Platform Command-line Interface (CLI), commonly known as **oc**.
- Log in as a user with **cluster-admin** privileges.

Procedure

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

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

- b. Create the namespace by running the following command:

```
$ 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:

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

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

```
$ 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.defaultChannel}'
```


4.3

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

Example Subscription

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

- ❶ Specify the value from you obtained in the previous step for the **.status.defaultChannel** parameter.

- ❷ You must specify the **redhat-operators** value.

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

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

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

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

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

8.2.1.2. Installing the Operator using the web console

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

**NOTE**

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

Procedure

1. Install the SR-IOV Network Operator using the OpenShift Container Platform web console:
 - a. In the OpenShift Container Platform web console, click **Operators → OperatorHub**.
 - b. Choose **SR-IOV Network Operator** from the list of available Operators, and then click **Install**.
 - c. On the **Create Operator Subscription** page, under **A specific namespace on the cluster** select **openshift-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 **openshift-sriov-network-operator** project with a **Status** of **InstallSucceeded**.



NOTE

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

If the operator does not appear as installed, to troubleshoot further:

- Go to the **Operators → Installed Operators** page and inspect the **Operator Subscriptions** and **Install Plans** tabs for any failure or errors under **Status**.
- Go to the **Workloads → Pods** page and check the logs for Pods in the **openshift-sriov-network-operator** project.

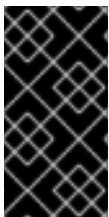
Next steps

- Optional: [Configuring the SR-IOV Network Operator](#)

8.3. CONFIGURING THE SR-IOV NETWORK OPERATOR

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

8.3.1. Configuring the SR-IOV Network Operator



IMPORTANT

Modifying the SR-IOV Network Operator configuration is not normally necessary. The default configuration is recommended for most use cases. Complete the steps to modify the relevant configuration only if the default behavior of the Operator is not compatible with your use case.

The SR-IOV Network Operator adds the **SriovOperatorConfig.sriovnetwork.openshift.io** CustomResourceDefinition resource. The operator automatically creates a SriovOperatorConfig custom resource (CR) named **default** in the **openshift-sriov-network-operator** namespace.



NOTE

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

The SriovOperatorConfig CR provides several fields for configuring the operator:

- **enableInjector** allows project administrators to enable or disable the Network Resources Injector DaemonSet.

- **enableOperatorWebhook** allows project administrators to enable or disable the Operator Admission Controller webhook DaemonSet.
- **configDaemonNodeSelector** allows project administrators to schedule the SR-IOV Network Config Daemon on selected nodes.

8.3.1.1. About the Network Resources Injector

The Network Resources Injector is a Kubernetes Dynamic Admission Controller application. It provides the following capabilities:

- Mutation of resource requests and limits in Pod specification to add an SR-IOV resource name according to an SR-IOV network attachment definition annotation.
- Mutation of Pod specifications with downward API volume to expose pod annotations and labels to the running container as files under the **/etc/podnetinfo** path.

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

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

NAME	READY	STATUS	RESTARTS	AGE
network-resources-injector-5cz5p	1/1	Running	0	10m
network-resources-injector-dwqpx	1/1	Running	0	10m
network-resources-injector-lktz5	1/1	Running	0	10m

8.3.1.2. About the SR-IOV Operator admission controller webhook

The SR-IOV Operator Admission Controller webhook is a Kubernetes Dynamic Admission Controller application. It provides the following capabilities:

- Validation of the **SriovNetworkNodePolicy** CR when it is created or updated.
- Mutation of the **SriovNetworkNodePolicy** CR by setting the default value for the **priority** and **deviceType** fields when the CR is created or updated.

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

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

NAME	READY	STATUS	RESTARTS	AGE
operator-webhook-9jkw6	1/1	Running	0	16m
operator-webhook-kbr5p	1/1	Running	0	16m
operator-webhook-rpfrl	1/1	Running	0	16m

8.3.1.3. About custom node selectors

The SR-IOV Network Config daemon discovers and configures the SR-IOV network devices on cluster nodes. By default, it is deployed to all the **worker** nodes in the cluster. You can use node labels to specify on which nodes the SR-IOV Network Config daemon runs.

8.3.1.4. Disabling or enabling the Network Resources Injector

To disable or enable the Network Resources Injector, which is enabled by default, complete the following procedure.

Prerequisites

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

Procedure

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

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

8.3.1.5. Disabling or enabling the SR-IOV Operator admission controller webhook

To disable or enable the admission controller webhook, which is enabled by default, complete the following procedure.

Prerequisites

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

Procedure

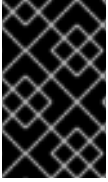
- Set the **enableOperatorWebhook** field. Replace **<value>** with **false** to disable the feature or **true** to enable it:

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

8.3.1.6. Configuring a custom NodeSelector for the SR-IOV Network Config daemon

The SR-IOV Network Config daemon discovers and configures the SR-IOV network devices on cluster nodes. By default, it is deployed to all the **worker** nodes in the cluster. You can use node labels to specify on which nodes the SR-IOV Network Config daemon runs.

To specify the nodes where the SR-IOV Network Config daemon is deployed, complete the following procedure.



IMPORTANT

When you update the **configDaemonNodeSelector** field, the SR-IOV Network Config daemon is recreated on each selected node. While the daemon is recreated, cluster users are unable to apply any new SR-IOV Network node policy or create new SR-IOV pods.

Procedure

- To update the node selector for the operator, enter the following command:

```
$ oc patch sriovoperatorconfig default --type=json \
-n openshift-sriov-network-operator \
--patch '{
  "op": "replace",
  "path": "/spec/configDaemonNodeSelector",
  "value": {<node-label>}
}'
```

Replace **<node-label>** with a label to apply as in the following example: **"node-role.kubernetes.io/worker": ""**.

Next steps

- [Configuring a SR-IOV network device](#)

8.4. CONFIGURING AN SR-IOV NETWORK DEVICE

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

8.4.1. Automated discovery of SR-IOV network devices

The SR-IOV Network Operator will search your cluster for SR-IOV capable network devices on worker nodes. The Operator creates and updates a `SriovNetworkNodeState` Custom Resource (CR) for each worker node that provides a compatible SR-IOV network device.

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



IMPORTANT

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

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

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodeState
metadata:
  name: node-25 1
  namespace: openshift-sriov-network-operator
ownerReferences:
- apiVersion: sriovnetwork.openshift.io/v1
  blockOwnerDeletion: true
```

```

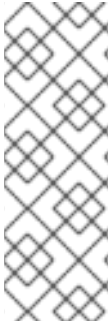
  controller: true
  kind: SriovNetworkNodePolicy
  name: default
spec:
  dpConfigVersion: "39824"
status:
  interfaces: 2
  - deviceID: "1017"
    driver: mlx5_core
    mtu: 1500
    name: ens785f0
    pciAddress: "0000:18:00.0"
    totalvfs: 8
    vendor: 15b3
  - deviceID: "1017"
    driver: mlx5_core
    mtu: 1500
    name: ens785f1
    pciAddress: "0000:18:00.1"
    totalvfs: 8
    vendor: 15b3
  - deviceID: 158b
    driver: i40e
    mtu: 1500
    name: ens817f0
    pciAddress: 0000:81:00.0
    totalvfs: 64
    vendor: "8086"
  - deviceID: 158b
    driver: i40e
    mtu: 1500
    name: ens817f1
    pciAddress: 0000:81:00.1
    totalvfs: 64
    vendor: "8086"
  - deviceID: 158b
    driver: i40e
    mtu: 1500
    name: ens803f0
    pciAddress: 0000:86:00.0
    totalvfs: 64
    vendor: "8086"
  syncStatus: Succeeded

```

- 1 The value for the **name** parameter is the same as the name of the worker node.
- 2 The **interfaces** collection includes a list of all of the SR-IOV devices discovered by the Operator on the worker node.

8.4.2. Configuring SR-IOV network devices

The SR-IOV Network Operator adds the **SriovNetworkNodePolicy.sriovnetwork.openshift.io** Custom Resource Definition (CRD) to OpenShift Container Platform. You can configure the SR-IOV network device by creating a SriovNetworkNodePolicy Custom Resource (CR).



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.

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SrioVNetworkNodePolicy
metadata:
  name: <name> 1
  namespace: openshift-sriov-network-operator 2
spec:
  resourceName: <sriov_resource_name> 3
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true" 4
  priority: <priority> 5
  mtu: <mtu> 6
  numVfs: <num> 7
  nicSelector: 8
    vendor: "<vendor_code>" 9
    deviceID: "<device_id>" 10
    pfNames: ["<pf_name>", ...] 11
    rootDevices: ["<pci_bus_id>", "..."] 12
  deviceType: <device_type> 13
  isRdma: false 14
```

- 1 Specify a name for the CR.
- 2 Specify the namespace where the SR-IOV Operator is installed.
- 3 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.
- 4 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.

- 5 Optional. 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**. The default value is **99**.
- 6 Optional. Specify a value for the maximum transmission unit (MTU) of the virtual function. The maximum MTU value can vary for different NIC models.
- 7 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**.
- 8 The **nicSelector** mapping selects the Ethernet device for the Operator to configure. You do not need to specify values for all the parameters. It is recommended to identify the Ethernet adapter with enough precision to minimize the possibility of selecting an Ethernet device unintentionally. If you specify **rootDevices**, you must also specify a value for **vendor**, **deviceID**, or **pfNames**. If you specify both **pfNames** and **rootDevices** at the same time, ensure that they point to an identical device.
- 9 Optional. Specify the vendor hex code of the SR-IOV network device. The only allowed values are either **8086** or **15b3**.
- 10 Optional. Specify the device hex code of SR-IOV network device. The only allowed values are **158b**, **1015**, **1017**.
- 11 Optional. The parameter accepts an array of one or more physical function (PF) names for the Ethernet device.
- 12 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**.
- 13 Optional. Specify the driver type for the virtual functions. You can specify one of the following values: **netdevice** or **vfiopci**. The default value is **netdevice**.



NOTE

For a Mellanox card to work in dpdk mode, use the netdevice driver type.

- 14 Optional. Specify whether to enable RDMA mode. The default value is **false**. Only RDMA over Converged Ethernet (RoCE) mode is supported on Mellanox Ethernet adapters.



NOTE

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> 1
```

- 1 Replace **<filename>** with the name of the file you created in the previous step.

Next steps

- [Configuring an SR-IOV network attachment](#)

8.5. CONFIGURING A SR-IOV NETWORK ATTACHMENT

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

8.5.1. Configuring SR-IOV additional network

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



NOTE

Do not modify or delete a SrioNetwork 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.

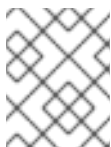
Procedure

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

```
apiVersion: srioNetwork.openshift.io/v1
kind: SrioNetwork
metadata:
  name: <name> 1
  namespace: openshift-srio-network-operator 2
spec:
  networkNamespace: <target_namespace> 3
  ipam: <ipam> 4
  vlan: <vlan> 5
  resourceName: <srio_resource_name> 6
  linkState: <link_state> 7
  maxTxRate: <max_tx_rate> 8
  minTxRate: <min_rx_rate> 9
  vlanQoS: <vlan_qos> 10
  spoofChk: "<spoof_check>" 11
  trust: "<trust_vf>" 12
  capabilities: <capabilities> 13
```

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 Optional. Replace **<target_namespace>** with the namespace where the NetworkAttachmentDefinition CR will be created. The default value is **openshift-sriov-network-operator**.
- 4 Optional. Replace **<ipam>** a configuration object for the ipam CNI plug-in as a YAML block scalar. The plug-in manages IP address assignment for the attachment definition.
- 5 Optional. Replace **<vlan>** with a Virtual LAN (VLAN) ID for the additional network. The integer value must be from **0** to **4095**. The default value is **0**.
- 6 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.
- 7 Optional. Replace **<link_state>** with the link state of Virtual Function (VF). Allowed value are **enable**, **disable** and **auto**.
- 8 Optional. Replace **<max_tx_rate>** with a maximum transmission rate, in Mbps, for the VF.
- 9 Optional. Replace **<min_tx_rate>** with a minimum transmission rate, in Mbps, for the VF. This value should always be less than or equal to Maximum transmission rate.

**NOTE**

Intel NICs do not support the **minTxRate** parameter. For more information, see [BZ#1772847](#).

- 10 Optional. Replace **<vlan_qos>** with an IEEE 802.1p priority level for the VF. The default value is **0**.
- 11 Optional. Replace **<spoof_check>** with the spoof check mode of the VF. The allowed values are the strings **"on"** and **"off"**.

**IMPORTANT**

You must enclose the value you specify in quotes or the CR will be rejected by the SR-IOV Network Operator.

- 12 Optional. Replace **<trust_vf>** with the trust mode of the VF. The allowed values are the strings **"on"** and **"off"**.

**IMPORTANT**

You must enclose the value you specify in quotes or the CR will be rejected by the SR-IOV Network Operator.

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

2. Create the CR by running the following command:

```
$ oc create -f <filename> 1
```

- 1 Replace **<filename>** with the name of the file you created in the previous step.

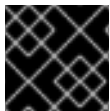
- Optional: Confirm that the NetworkAttachmentDefinition CR associated with the SrioNetwork CR that you created in the previous step exists by running the following command. Replace **<namespace>** with the namespace you specified in the SrioNetwork CR.

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

8.5.1.1. 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.

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>", 1
    "addresses": [ 2
      {
        "address": "<address>", 3
        "gateway": "<gateway>" 4
      }
    ],
    "routes": [ 5
      {
        "dst": "<dst>" 6
        "gw": "<gw>" 7
      }
    ],
    "dns": { 8
      "nameservers": ["<nameserver>"], 9
      "domain": "<domain>", 10
      "search": ["<search_domain>"] 11
    }
  }
}
```

- 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**.

- 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 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.

8.5.1.1.1. Static IP address assignment configuration example

You can configure ipam for static IP address assignment:

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

8.5.1.1.2. Dynamic IP address assignment configuration example

You can configure ipam for DHCP:

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

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

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

Prerequisites

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

Procedure

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

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SrioNetwork
metadata:
  name: <name> ❶
  namespace: srio-network-operator ❷
spec:
  networkNamespace: <target_namespace> ❸
  ipam: '{"type": "static"}' ❹
  capabilities: '{"mac": true, "ips": true}' ❺
  resourceName: <srio_resource_name> ❻
```

- ❶ Replace **<name>** with a name for the CR. The Operator will create a NetworkAttachmentDefinition CR with same name.
- ❷ Specify the namespace where the SR-IOV Operator is installed.
- ❸ Replace **<target_namespace>** with the namespace where the NetworkAttachmentDefinition CR will be created.
- ❹ Specify static type for the ipam CNI plug-in as a YAML block scalar.
- ❺ Specify **mac** and **ips capabilities** to **true**.
- ❻ Replace **<srio_resource_name>** with the value for the **.spec.resourceName** parameter from the SrioNetworkNodePolicy CR that defines the SR-IOV hardware for this additional network.

2. Create the CR by running the following command:

```
$ oc create -f <filename> ❶
```

- ❶ Replace **<filename>** with the name of the file you created in the previous step.
3. Optional: Confirm that the NetworkAttachmentDefinition CR associated with the SrioNetwork CR that you created in the previous step exists by running the following command. Replace **<namespace>** with the namespace you specified in the SrioNetwork CR.

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



NOTE

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

1. Create the following SR-IOV pod spec, and then save the YAML in the **<name>-srio-pod.yaml** file. Replace **<name>** with a name for this pod.

```

apiVersion: v1
kind: Pod
metadata:
  name: sample-pod
  annotations:
    k8s.v1.cni.cncf.io/networks: '[
  {
    "name": "<name>", ❶
    "mac": "20:04:0f:f1:88:01", ❷
    "ips": ["192.168.10.1/24", "2001::1/64"] ❸
  }
]'
spec:
  containers:
  - name: sample-container
    image: <image>
    imagePullPolicy: IfNotPresent
    command: ["sleep", "infinity"]

```

- ❶ Replace **<name>** with the name of the SR-IOV network attachment definition CR.
- ❷ Specify the **mac** address for the SR-IOV device which is allocated from the resource type defined in the SR-IOV network attachment definition CR.
- ❸ Specify the IPv4 and/or IPv6 addresses for the SR-IOV device which is allocated from the resource type defined in the SR-IOV network attachment definition CR.

2. Create the sample SR-IOV pod by running the following command:

```
$ oc create -f <filename> ❶
```

- ❶ Replace **<filename>** with the name of the file you created in the previous step.
3. Optional: Confirm that **mac** and **ips** addresses are applied to the SR-IOV device by running the following command. Replace **<namespace>** with the namespace you specified in the SrioNetwork CR.

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

Next steps

- [Adding a Pod to an SR-IOV additional network](#)

8.6. ADDING A POD TO AN SR-IOV ADDITIONAL NETWORK

You can add a Pod to an existing Single Root I/O Virtualization (SR-IOV) network.

8.6.1. Adding a Pod to an additional network

You can add a Pod to an additional network. The Pod continues to send normal cluster related network traffic over the default network.



NOTE

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

Prerequisites

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

Procedure

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

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

```
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: <network>[,<network>,...] 1
```

- 1 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:

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

2. Create the Pod by running the following command:

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

3. Optional: Confirm that the annotation exists in the Pod CR by running the following command. Replace **<name>** with the name of the Pod.

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

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

```
$ oc get pod example-pod -o yaml
apiVersion: v1
kind: Pod
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: macvlan-bridge
    k8s.v1.cni.cncf.io/networks-status: |- 1
    [{
      "name": "openshift-sdn",
      "interface": "eth0",
      "ips": [
        "10.128.2.14"
      ],
      "default": true,
      "dns": {}
    },{
      "name": "macvlan-bridge",
      "interface": "net1",
      "ips": [
        "20.2.2.100"
      ],
      "mac": "22:2f:60:a5:f8:00",
      "dns": {}
    }]
  name: example-pod
  namespace: default
spec:
  ...
status:
  ...
```

- 1** The **k8s.v1.cni.cncf.io/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.

8.6.2. Creating a non-uniform memory access (NUMA) aligned SR-IOV pod

You can create a NUMA aligned SR-IOV pod by restricting SR-IOV and the CPU resources allocated from the same NUMA node with **restricted** or **single-numa-node** Topology Manager policies.

Prerequisites

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

Procedure

1. Create the following SR-IOV pod spec, and then save the YAML in the **<name>-sriov-pod.yaml** file. Replace **<name>** with a name for this pod.

The following example shows a SR-IOV pod spec:

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

- ❶ Replace **<name>** with the name of the SR-IOV network attachment definition CR.
- ❷ Replace **<image>** with the name of the **sample-pod** image.
- ❸ To create the SR-IOV pod with guaranteed QoS, set **memory limits** equal to **memory requests**.
- ❹ To create the SR-IOV pod with guaranteed QoS, set **cpu limits** equals to **cpu requests**.

2. Create the sample SR-IOV pod by running the following command:

```
$ oc create -f <filename> ❶
```

- ❶ Replace **<filename>** with the name of the file you created in the previous step.

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

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

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

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

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

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

8.7. USING HIGH PERFORMANCE MULTICAST

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

8.7.1. Configuring high performance multicast

OpenShift SDN supports multicast between Pods on the default network. This is best used for low-bandwidth coordination or service discovery, and not a high-bandwidth solution. For applications of streaming media, such as IPTV and multipoint videoconferencing, you can utilize SR-IOV to provide near-native performance.

When using additional SR-IOV interfaces for multicast:

- Multicast packages must be sent or received by a Pod through the additional SR-IOV interface.
- The physical network which connects the SR-IOV interfaces decides the multicast routing and topology, which is not controlled by OpenShift.

8.7.2. Using an SR-IOV interface for multicast

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

Prerequisites

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

Procedure

1. Create a SrioNetworkNodePolicy CR:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SrioNetworkNodePolicy
metadata:
  name: policy-example
  namespace: openshift-sriov-network-operator
spec:
  resourceName: example
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  numVfs: 4
  nicSelector:
    vendor: "8086"
    pfNames: ['ens803f0']
    rootDevices: ['0000:86:00.0']
```

2. Create a SrioNetwork CR:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SrioNetwork
metadata:
  name: net-example
  namespace: openshift-sriov-network-operator
spec:
```

```

networkNamespace: default
ipam: |
{
  "type": "host-local", ❶
  "subnet": "10.56.217.0/24",
  "rangeStart": "10.56.217.171",
  "rangeEnd": "10.56.217.181",
  "routes": [
    {"dst": "224.0.0.0/5"},
    {"dst": "232.0.0.0/5"}
  ],
  "gateway": "10.56.217.1"
}
resourceName: example

```

- ❶ Make sure to provision the following default routes through your DHCP server 224.0.0.0/5, 232.0.0.0/5, if you choose to use DHCP as IPAM. This is to override the static multicast route set by Openshift SDN.

3. Create a Pod with multicast application:

```

apiVersion: v1
kind: Pod
metadata:
  name: testpmd
  namespace: default
  annotations:
    k8s.v1.cni.cncf.io/networks: nic1
spec:
  containers:
  - name: example
    image: rhel7:latest
    securityContext:
      capabilities:
        add: ["NET_ADMIN"] ❶
    command: [ "sleep", "infinity" ]

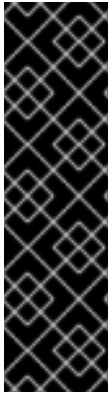
```

- ❶ The NET_ADMIN capability is only required if your application needs to assign the multicast IP address to the SR-IOV interface. Otherwise, it can be omitted.

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

You can use Single Root I/O Virtualization (SR-IOV) network hardware with the Data Plane Development Kit (DPDK) and with remote direct memory access (RDMA).

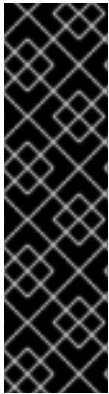
8.8.1. Examples of using virtual functions in DPDK and RDMA modes



IMPORTANT

The Data Plane Development Kit (DPDK) is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see <https://access.redhat.com/support/offerings/techpreview/>.



IMPORTANT

Remote Direct Memory Access (RDMA) is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see <https://access.redhat.com/support/offerings/techpreview/>.

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.

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

Procedure

1. Create the following SrioNetworkNodePolicy CR, and then save the YAML in the **intel-dpdk-node-policy.yaml** file.

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SrioNetworkNodePolicy
metadata:
  name: intel-dpdk-node-policy
  namespace: openshift-sriov-network-operator
spec:
  resourceName: intelnics
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  priority: <priority>
  numVfs: <num>
  nicSelector:
    vendor: "8086"
    deviceID: "158b"
```

```
pfNames: ["<pf_name>", ...]
rootDevices: ["<pci_bus_id>", "..."]
deviceType: vfio-pci 1
```

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



NOTE

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

+ When applying the configuration specified in a SriovNetworkNodePolicy CR, the SR-IOV Operator may drain the nodes, and in some cases, reboot nodes. It may take several minutes for a configuration change to apply. Ensure that there are enough available nodes in your cluster to handle the evicted workload beforehand.

+ After the configuration update is applied, all the Pods in **openshift-sriov-network-operator** namespace will change to a **Running** status.

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

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

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

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: intel-dpdk-network
  namespace: openshift-sriov-network-operator
spec:
  networkNamespace: <target_namespace>
  ipam: "{}" 1
  vlan: <vlan>
  resourceName: intelNic
```

- 1** Specify an empty object "{}" for the ipam CNI plug-in. DPDK works in userspace mode and does not require an IP address.



NOTE

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

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

```
$ oc create -f intel-dpdk-network.yaml
```

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

—

```

apiVersion: v1
kind: Pod
metadata:
  name: dpdk-app
  namespace: <target_namespace> ❶
  annotations:
    k8s.v1.cni.cncf.io/networks: intel-dpdk-network
spec:
  containers:
    - name: testpmd
      image: <DPDK_image> ❷
      securityContext:
        capabilities:
          add: ["IPC_LOCK"] ❸
      volumeMounts:
        - mountPath: /dev/hugepages ❹
          name: hugepage
      resources:
        limits:
          openshift.io/intelnic: "1" ❺
          memory: "1Gi"
          cpu: "4" ❻
          hugepages-1Gi: "4Gi" ❼
        requests:
          openshift.io/intelnic: "1"
          memory: "1Gi"
          cpu: "4"
          hugepages-1Gi: "4Gi"
      command: ["sleep", "infinity"]
  volumes:
    - name: hugepage
      emptyDir:
        medium: HugePages

```

- ❶ Specify the same **target_namespace** where the SrioNetwork CR **intel-dpdk-network** is created. If you would like to create the Pod in a different namespace, change **target_namespace** in both the Pod spec and the SrioNetwork CR.
- ❷ Specify the DPDK image which includes your application and the DPDK library used by application.
- ❸ Specify the **IPC_LOCK** capability which is required by the application to allocate hugepage memory inside container.
- ❹ Mount a hugepage volume to the DPDK Pod under **/dev/hugepages**. The hugepage volume is backed by the emptyDir volume type with the medium being **HugePages**.
- ❺ (optional) Specify the number of DPDK devices allocated to DPDK Pod. This resource request and limit, if not explicitly specified, will be automatically added by the SR-IOV network resource injector. The SR-IOV network resource injector is an admission controller component managed by the SR-IOV Operator. It is enabled by default and can be disabled by setting **Injector** option to **false** in the default **SrioOperatorConfig** CR.
- ❻ Specify the number of CPUs. The DPDK Pod usually requires exclusive CPUs to be allocated from the kubelet. This is achieved by setting CPU Manager policy to **static** and creating a Pod with **Guaranteed** QoS.

creating a Pod with **guaranteed QoS**.

- 7 Specify hugepage size **hugepages-1Gi** or **hugepages-2Mi** and the quantity of hugepages that will be allocated to the DPDK Pod. Configure **2Mi** and **1Gi** hugepages separately. Configuring **1Gi** hugepage requires adding kernel arguments to Nodes. For example, adding kernel arguments **default_hugepagesz=1GB**, **hugepagesz=1G** and **hugepages=16** will result in **16*1Gi** hugepages be allocated during system boot.

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

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

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

Procedure

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

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SrivNetworkNodePolicy
metadata:
  name: mlx-dpdk-node-policy
  namespace: openshift-sriov-network-operator
spec:
  resourceName: mlxnic
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  priority: <priority>
  numVfs: <num>
  nicSelector:
    vendor: "15b3"
    deviceID: "1015" 1
    pfNames: ["<pf_name>", ...]
    rootDevices: ["<pci_bus_id>", "..."]
  deviceType: netdevice 2
  isRdma: true 3
```

- 1 Specify the device hex code of the SR-IOV network device. The only allowed values for Mellanox cards are **1015**, **1017**.
- 2 Specify the driver type for the virtual functions to **netdevice**. Mellanox SR-IOV VF can work in DPDK mode without using the **vfio-pci** device type. VF device appears as a kernel network interface inside a container.
- 3 Enable RDMA mode. This is required by Mellanox cards to work in DPDK mode.

**NOTE**

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

+ When applying the configuration specified in a SriovNetworkNodePolicy CR, the SR-IOV Operator may drain the nodes, and in some cases, reboot nodes. It may take several minutes for a configuration change to apply. Ensure that there are enough available nodes in your cluster to handle the evicted workload beforehand.

+ After the configuration update is applied, all the Pods in the **openshift-sriov-network-operator** namespace will change to a **Running** status.

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

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

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

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: mlx-dpdk-network
  namespace: openshift-sriov-network-operator
spec:
  networkNamespace: <target_namespace>
  ipam: |- 1
  ...
  vlan: <vlan>
  resourceName: mlxnic
```

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

**NOTE**

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

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

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

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

```
apiVersion: v1
kind: Pod
metadata:
  name: dpdk-app
  namespace: <target_namespace> 1
  annotations:
```



```

k8s.v1.cni.cncf.io/networks: mlx-dpdk-network
spec:
  containers:
  - name: testpmd
    image: <DPDK_image> ❷
    securityContext:
      capabilities:
        add: ["IPC_LOCK"] ❸
    volumeMounts:
    - mountPath: /dev/hugepages ❹
      name: hugepage
  resources:
    limits:
      openshift.io/mlxnics: "1" ❺
      memory: "1Gi"
      cpu: "4" ❻
      hugepages-1Gi: "4Gi" ❼
    requests:
      openshift.io/mlxnics: "1"
      memory: "1Gi"
      cpu: "4"
      hugepages-1Gi: "4Gi"
    command: ["sleep", "infinity"]
  volumes:
  - name: hugepage
    emptyDir:
      medium: HugePages

```

- ❶ Specify the same **target_namespace** where SrivNetwork CR **mlx-dpdk-network** is created. If you would like to create the Pod in a different namespace, change **target_namespace** in both Pod spec and SrivNetowrk CR.
- ❷ Specify the DPDK image which includes your application and the DPDK library used by application.
- ❸ Specify the **IPC_LOCK** capability which is required by the application to allocate hugepage memory inside the container.
- ❹ Mount the hugepage volume to the DPDK Pod under **/dev/hugepages**. The hugepage volume is backed by the emptyDir volume type with the medium being **Hugepages**.
- ❺ (optional) Specify the number of DPDK devices allocated to the DPDK Pod. This resource request and limit, if not explicitly specified, will be automatically added by SR-IOV network resource injector. The SR-IOV network resource injector is an admission controller component managed by SR-IOV Operator. It is enabled by default and can be disabled by setting the **Injector** option to **false** in the default **SrivOperatorConfig** CR.
- ❻ Specify the number of CPUs. The DPDK Pod usually requires exclusive CPUs be allocated from kubelet. This is achieved by setting CPU Manager policy to **static** and creating a Pod with **Guaranteed** QoS.
- ❼ Specify hugepage size **hugepages-1Gi** or **hugepages-2Mi** and the quantity of hugepages that will be allocated to DPDK Pod. Configure **2Mi** and **1Gi** hugepages separately. Configuring **1Gi** hugepage requires adding kernel arguments to Nodes.

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

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

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

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

Procedure

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

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SrioVNetworkNodePolicy
metadata:
  name: mlx-rdma-node-policy
  namespace: openshift-sriov-network-operator
spec:
  resourceName: mlxnic
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  priority: <priority>
  numVfs: <num>
  nicSelector:
    vendor: "15b3"
    deviceID: "1015" ❶
    pfNames: ["<pf_name>", ...]
    rootDevices: ["<pci_bus_id>", "..."]
  deviceType: netdevice ❷
  isRdma: true ❸
```

- ❶ Specify the device hex code of SR-IOV network device. The only allowed values for Mellanox cards are **1015**, **1017**.
- ❷ Specify the driver type for the virtual functions to **netdevice**.
- ❸ Enable RDMA mode.



NOTE

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

+ When applying the configuration specified in a SrioVNetworkNodePolicy CR, the SR-IOV Operator may drain the nodes, and in some cases, reboot nodes. It may take several minutes for a configuration change to apply. Ensure that there are enough available nodes in your cluster to handle the evicted workload beforehand.

+ After the configuration update is applied, all the Pods in the **openshift-sriov-network-operator** namespace will change to a **Running** status.

2. Create the SrioNetworkNodePolicy CR by running the following command:

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

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

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

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



NOTE

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

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

```
$ oc create -f mlx-rdma-network.yaml
```

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

```
apiVersion: v1
kind: Pod
metadata:
  name: rdma-app
  namespace: <target_namespace> ❶
  annotations:
    k8s.v1.cni.cncf.io/networks: mlx-rdma-network
spec:
  containers:
    - name: testpmd
      image: <RDMA_image> ❷
      securityContext:
        capabilities:
          add: ["IPC_LOCK"] ❸
      volumeMounts:
        - mountPath: /dev/hugepages ❹
          name: hugepage
  resources:
    limits:
```

```

    memory: "1Gi"
    cpu: "4" 5
    hugepages-1Gi: "4Gi" 6
  requests:
    memory: "1Gi"
    cpu: "4"
    hugepages-1Gi: "4Gi"
  command: ["sleep", "infinity"]
  volumes:
  - name: hugepage
    emptyDir:
      medium: HugePages

```

- 1** Specify the same **target_namespace** where SrioNetwork CR **mlx-rdma-network** is created. If you would like to create the Pod in a different namespace, change **target_namespace** in both Pod spec and SrioNetwork CR.
- 2** Specify the RDMA image which includes your application and RDMA library used by application.
- 3** Specify the **IPC_LOCK** capability which is required by the application to allocate hugepage memory inside the container.
- 4** Mount the hugepage volume to RDMA Pod under **/dev/hugepages**. The hugepage volume is backed by the emptyDir volume type with the medium being **Hugepages**.
- 5** Specify number of CPUs. The RDMA Pod usually requires exclusive CPUs be allocated from the kubelet. This is achieved by setting CPU Manager policy to **static** and create Pod with **Guaranteed** QoS.
- 6** Specify hugepage size **hugepages-1Gi** or **hugepages-2Mi** and the quantity of hugepages that will be allocated to the RDMA Pod. Configure **2Mi** and **1Gi** hugepages separately. Configuring **1Gi** hugepage requires adding kernel arguments to Nodes.

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

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

CHAPTER 9. OPENSIFT SDN NETWORK PROVIDER

9.1. ABOUT OPENSIFT 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 4.3.
- 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.

9.2. CONFIGURING EGRESS IPS FOR A PROJECT

As a cluster administrator, you can configure the OpenShift SDN network provider to assign one or more egress IP addresses to a project.

9.2.1. Egress IP address assignment for project egress traffic

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

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



IMPORTANT

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

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

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

- In the *automatically assigned* approach, an egress IP address range is assigned to a node.

- In the *manually assigned* approach, a list of one or more egress IP address is assigned to a node.

Namespaces that request an egress IP address are matched with nodes that can host those egress IP addresses, and then the egress IP addresses are assigned to those nodes. If the **egressIPs** parameter is set on a **NetNamespace** resource, but no node hosts that egress IP address, then egress traffic from the namespace will be dropped.

High availability of nodes is automatic. If a node that hosts an egress IP address is unreachable and there are nodes that are able to host that egress IP address, then the egress IP address will move to a new node. When the unreachable node comes back online, the egress IP address automatically moves to balance egress IP addresses across nodes.



IMPORTANT

You cannot use manually assigned and automatically assigned egress IP addresses on the same nodes. If you manually assign egress IP addresses from an IP address range, you must not make that range available for automatic IP assignment.

9.2.1.1. Considerations when using automatically assigned egress IP addresses

When using the automatic assignment approach for egress IP addresses the following considerations apply:

- You set the **egressCIDRs** parameter of each node's **HostSubnet** resource to indicate the range of egress IP addresses that can be hosted by a node. OpenShift Container Platform sets the **egressIPs** parameter of the **HostSubnet** resource based on the IP address range you specify.
- Only a single egress IP address per namespace is supported when using the automatic assignment mode.

If the node hosting the namespace's egress IP address is unreachable, OpenShift Container Platform will reassign the egress IP address to another node with a compatible egress IP address range. The automatic assignment approach works best for clusters installed in environments with flexibility in associating additional IP addresses with nodes.

9.2.1.2. Considerations when using manually assigned egress IP addresses

When using the manual assignment approach for egress IP addresses the following considerations apply:

- You set the **egressIPs** parameter of each node's **HostSubnet** resource to indicate the IP addresses that can be hosted by a node.
- Multiple egress IP addresses per namespace are supported.

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

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

9.2.2. Configuring automatically assigned egress IP addresses for a namespace

In OpenShift Container Platform you can enable automatic assignment of an egress IP address for a specific namespace across one or more nodes.

Prerequisites

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

Procedure

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

```
$ oc patch netnamespace <project_name> --type=merge -p \ 1
{
  "egressIPs": [
    "<ip_address>" 2
  ]
}
```

- 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:

```
$ oc patch hostsubnet <node_name> --type=merge -p \ 1
{
  "egressCIDRs": [
    "<ip_address_range_1>", "<ip_address_range_2>" 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:

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

OpenShift Container Platform automatically assigns specific egress IP addresses to available nodes in a balanced way. In this case, it assigns the egress IP address 192.168.1.100 to **node1** and the egress IP address 192.168.1.101 to **node2** or vice versa.

9.2.3. Configuring manually assigned egress IP addresses for a namespace

In OpenShift Container Platform you can associate one or more egress IP addresses with a namespace.

Prerequisites

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

Procedure

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

```
$ oc patch netnamespace <project> --type=merge -p \ 1
{'
  "egressIPs": [ 2
    "<ip_address>"
  ]
}'
```

- 1** Specify the name of the project.
- 2** Specify one or more egress IP addresses. The **egressIPs** parameter is an array.

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

```
$ 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 \ 1
{'
  "egressIPs": [ 2
    "<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.

9.3. CONFIGURING AN EGRESS FIREWALL TO CONTROL ACCESS TO EXTERNAL IP ADDRESSES

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

9.3.1. How an egress firewall works in a project

As a cluster administrator, you can use an *egress firewall* to limit the external hosts that some or all Pods can access from within the cluster. An egress firewall supports the following scenarios:

- A Pod can only connect to internal hosts and cannot initiate connections to the public Internet.
- A Pod can only connect to the public Internet and cannot initiate connections to internal hosts that are outside the OpenShift Container Platform cluster.
- A Pod cannot reach specified internal subnets or hosts outside the OpenShift Container Platform cluster.
- A Pod can connect to only specific external hosts.

You configure an egress firewall policy by creating an EgressNetworkPolicy Custom Resource (CR) object and specifying an IP address range in CIDR format or by specifying a DNS name. For example, you can allow one project access to a specified IP range but deny the same access to a different project. Or you can restrict application developers from updating from Python pip mirrors, and force updates to come only from approved sources.



IMPORTANT

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

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

CAUTION

Egress firewall rules do not apply to traffic that goes through routers. Any user with permission to create a Route CR object can bypass egress network policy rules by creating a route that points to a forbidden destination.

9.3.1.1. Limitations of an egress firewall

An egress firewall has the following limitations:

- No project can have more than one EgressNetworkPolicy object.
- The **default** project cannot use egress network policy.
- When using the OpenShift SDN network provider in multitenant mode, the following limitations apply:
 - Global projects cannot use an egress firewall. You can make a project global by using the **oc adm pod-network make-projects-global** command.
 - Projects merged by using the **oc adm pod-network join-projects** command cannot use an egress firewall in any of the joined projects.

Violating any of these restrictions results in broken egress network policy for the project, and may cause all external network traffic to be dropped.

9.3.1.2. Matching order for egress network policy rules

The egress network policy rules are evaluated in the order that they are defined, from first to last. The first rule that matches an egress connection from a Pod applies. Any subsequent rules are ignored for that connection.

9.3.1.3. How Domain Name Server (DNS) resolution works

If you use DNS names in any of your egress firewall policy rules, proper resolution of the domain names is subject to the following restrictions:

- Domain name updates are polled based on the TTL (time to live) value of the domain returned by the local non-authoritative servers.
- The Pod must resolve the domain from the same local name servers when necessary. Otherwise the IP addresses for the domain known by the egress firewall controller and the Pod can be different. If the IP addresses for a host name differ, the egress firewall might not be enforced consistently.
- Because the egress firewall controller and Pods asynchronously poll the same local name server, the Pod might obtain the updated IP address before the egress controller does, which causes a race condition. Due to this current limitation, domain name usage in EgressNetworkPolicy objects is only recommended for domains with infrequent IP address changes.



NOTE

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

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

9.3.2. EgressNetworkPolicy custom resource (CR) object

The following YAML describes an EgressNetworkPolicy CR object:

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

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

9.3.2.1. EgressNetworkPolicy rules

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

```
egress:
- type: <type> ❶
  to: ❷
    cidrSelector: <cidr> ❸
    dnsName: <dns-name> ❹
```

- ❶ Specify the type of rule. The value must be either **Allow** or **Deny**.
- ❷ Specify a value for either the **cidrSelector** key or the **dnsName** key for the rule. You cannot use both keys in a rule.
- ❸ Specify an IP address range in CIDR format.
- ❹ Specify a domain name.

9.3.2.2. Example EgressNetworkPolicy CR object

The following example defines several egress firewall policy rules:

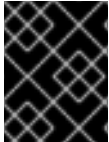
```
kind: EgressNetworkPolicy
apiVersion: v1
metadata:
  name: default-rules ❶
spec:
  egress: ❷
  - type: Allow
    to:
      cidrSelector: 1.2.3.0/24
  - type: Allow
    to:
      dnsName: www.example.com
```

```
- type: Deny
  to:
    cidrSelector: 0.0.0.0/0
```

- 1 The name for the policy object.
- 2 A collection of egress firewall policy rule objects.

9.3.3. Creating an egress firewall policy object

As a cluster administrator, you can create an egress firewall policy object for a project.



IMPORTANT

If the project already has an EgressNetworkPolicy object defined, you must edit the existing policy to make changes to the egress firewall rules.

Prerequisites

- A cluster that uses the OpenShift SDN network provider plug-in.
- Install the OpenShift Command-line Interface (CLI), commonly known as **oc**.
- You must log in to the cluster as a cluster administrator.

Procedure

1. Create a policy rule:
 - a. Create a **<policy-name>.yaml** file where **<policy-name>** describes the egress policy rules.
 - b. In the file you created, define an egress policy object.
2. Enter the following command to create the policy object:

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

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

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

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

9.4. EDITING AN EGRESS FIREWALL FOR A PROJECT

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

9.4.1. Editing an EgressNetworkPolicy object

As a cluster administrator, you can update the egress firewall for a project.

Prerequisites

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

Procedure

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

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

```
$ oc get -n <project> egressnetworkpolicy
```

2. Optionally, if you did not save a copy of the EgressNetworkPolicy object when you created the egress network firewall, enter the following command to create a copy.

```
$ oc get -n <project> \ 1
  egressnetworkpolicy <name> \ 2
  -o yaml > <filename>.yaml 3
```

1 Replace **<project>** with the name of the project

2 Replace **<name>** with the name of the object.

3 Replace **<filename>** with the name of the file to save the YAML.

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

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

9.4.2. EgressNetworkPolicy custom resource (CR) object

The following YAML describes an EgressNetworkPolicy CR object:

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

1 Specify a **name** for your egress firewall policy.

2 Specify a collection of one or more egress network policy rules as described in the following section.

9.4.2.1. EgressNetworkPolicy rules

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

```
egress:
- type: <type> ❶
  to: ❷
    cidrSelector: <cidr> ❸
    dnsName: <dns-name> ❹
```

- ❶ Specify the type of rule. The value must be either **Allow** or **Deny**.
- ❷ Specify a value for either the **cidrSelector** key or the **dnsName** key for the rule. You cannot use both keys in a rule.
- ❸ Specify an IP address range in CIDR format.
- ❹ Specify a domain name.

9.4.2.2. Example EgressNetworkPolicy CR object

The following example defines several egress firewall policy rules:

```
kind: EgressNetworkPolicy
apiVersion: v1
metadata:
  name: default-rules ❶
spec:
  egress: ❷
  - type: Allow
    to:
      cidrSelector: 1.2.3.0/24
  - type: Allow
    to:
      dnsName: www.example.com
  - type: Deny
    to:
      cidrSelector: 0.0.0.0/0
```

- ❶ The name for the policy object.
- ❷ A collection of egress firewall policy rule objects.

9.5. REMOVING AN EGRESS FIREWALL FROM A PROJECT

As a cluster administrator, you can remove an egress firewall from a project to remove all restrictions on network traffic from the project that leaves the OpenShift Container Platform cluster.

9.5.1. Removing an EgressNetworkPolicy object

As a cluster administrator, you can remove an egress firewall from a project.

Prerequisites

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

Procedure

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

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

```
$ oc get -n <project> egressnetworkpolicy
```

2. Enter the following command to delete the EgressNetworkPolicy object. Replace **<project>** with the name of the project and **<name>** with the name of the object.

```
$ oc delete -n <project> egressnetworkpolicy <name>
```

9.6. USING MULTICAST

9.6.1. About multicast

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



IMPORTANT

At this time, multicast is best used for low-bandwidth coordination or service discovery and not a high-bandwidth solution.

Multicast traffic between OpenShift Container Platform Pods is disabled by default. If you are using the OpenShift SDN 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.

9.6.2. Enabling multicast between Pods

You can enable multicast between Pods for your project.

Prerequisites

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

Procedure

- Run the following command to enable multicast for a project:

```
$ oc annotate netnamespace <namespace> \ 1  
    netnamespace.network.openshift.io/multicast-enabled=true
```

- 1** The **namespace** for the project you want to enable multicast for.

9.6.3. 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> \ 1  
    netnamespace.network.openshift.io/multicast-enabled-
```

- 1** The **namespace** for the project you want to disable multicast for.

9.7. CONFIGURING NETWORK ISOLATION USING OPENSIFT SDN

When your cluster is configured to use the multitenant isolation mode for the OpenShift SDN CNI plugin, each project is isolated by default. Network traffic is not allowed between Pods or services in different projects in multitenant isolation mode.

You can change the behavior of multitenant isolation for a project in two ways:

- You can join one or more projects, allowing network traffic between Pods and services in different projects.
- You can disable network isolation for a project. It will be globally accessible, accepting network traffic from Pods and services in all other projects. A globally accessible project can access Pods and services in all other projects.

Prerequisites

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

9.7.1. 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.

9.7.2. 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:

```
$ 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.

9.7.3. Disabling network isolation for a project

You can disable network isolation for a project.

Prerequisites

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

Procedure

- Run the following command for the project:

```
$ oc adm pod-network make-projects-global <project1> <project2>
```

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

9.8. 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.

9.8.1. 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.

9.8.2. 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.

9.8.3. kube-proxy configuration parameters

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

Table 9.1. Parameters

Parameter	Description	Values	Default
iptablesSyncPeriod	The refresh period for iptables rules.	A time interval, such as 30s or 2m . Valid suffixes include s , m , and h and are described in the Go time package documentation.	30s
proxyArguments.iptables-min-sync-period	The minimum duration before refreshing iptables rules. This parameter ensures that the refresh does not happen too frequently.	A time interval, such as 30s or 2m . Valid suffixes include s , m , and h and are described in the Go time package	30s

CHAPTER 10. CONFIGURING ROUTES

10.1. ROUTE CONFIGURATION

10.1.1. 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> 1
```

- 1 Supported time units are microseconds (us), milliseconds (ms), seconds (s), minutes (m), hours (h), or days (d).

The following example sets a timeout of two seconds on a route named **myroute**:

```
$ oc annotate route myroute --overwrite haproxy.router.openshift.io/timeout=2s
```

10.1.2. 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
```

```
kind: Route
metadata:
  annotations:
    haproxy.router.openshift.io/hsts_header: max-age=31536000;includeSubDomains;preload
```

1 2 3

- 1 **max-age** is the only required parameter. It measures the length of time, in seconds, that the HSTS policy is in effect. The client updates **max-age** whenever a response with a HSTS header is received from the host. When **max-age** times out, the client discards the policy.
- 2 **includeSubDomains** is optional. When included, it tells the client that all subdomains of the host are to be treated the same as the host.
- 3 **preload** is optional. When **max-age** is greater than 0, then including **preload** in **haproxy.router.openshift.io/hsts_header** allows external services to include this site in their HSTS preload lists. For example, sites such as Google can construct a list of sites that have **preload** set. Browsers can then use these lists to determine which sites they can communicate with over HTTPS, before they have interacted with the site. Without **preload** set, browsers must have interacted with the site over HTTPS to get the header.

10.1.3. 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 **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](#).

10.1.4. 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.

10.1.4.1. 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
```

10.1.5. Route-specific annotations

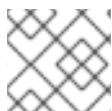
The Ingress Controller can set the default options for all the routes it exposes. An individual route can override some of these defaults by providing specific configurations in its annotations.

Table 10.1. Route annotations

Variable	Description	Environment variable used as default
haproxy.router.openshift.io/balance	Sets the load-balancing algorithm. Available options are source , roundrobin , and leastconn .	ROUTER_TCP_BALANCE_SCHEME for passthrough routes. Otherwise, use ROUTER_LOAD_BALANCE_ALGORITHM .

Variable	Description	Environment variable used as default
haproxy.router.openshift.io/disable_cookies	Disables the use of cookies to track related connections. If set to true or TRUE , the balance algorithm is used to choose which back-end serves connections for each incoming HTTP request.	
router.openshift.io/cookie_name	Specifies an optional cookie to use for this route. The name must consist of any combination of upper and lower case letters, digits, "_", and "-". The default is the hashed internal key name for the route.	
haproxy.router.openshift.io/pod-concurrent-connections	Sets the maximum number of connections that are allowed to a backing pod from a router. Note: if there are multiple pods, each can have this many connections. But if you have multiple routers, there is no coordination among them, each may connect this many times. If not set, or set to 0, there is no limit.	
haproxy.router.openshift.io/rate-limit-connections	Setting true or TRUE to enables rate limiting functionality.	
haproxy.router.openshift.io/rate-limit-connections.concurrent-tcp	Limits the number of concurrent TCP connections shared by an IP address.	
haproxy.router.openshift.io/rate-limit-connections.rate-http	Limits the rate at which an IP address can make HTTP requests.	
haproxy.router.openshift.io/rate-limit-connections.rate-tcp	Limits the rate at which an IP address can make TCP connections.	
haproxy.router.openshift.io/timeout	Sets a server-side timeout for the route. (TimeUnits)	ROUTER_DEFAULT_SERVER_TIMEOUT
router.openshift.io/haproxy.health.check.interval	Sets the interval for the back-end health checks. (TimeUnits)	ROUTER_BACKEND_CHECK_INTERVAL

Variable	Description	Environment variable used as default
<code>haproxy.router.openshift.io/iptables_whitelist</code>	Sets a whitelist for the route.	
<code>haproxy.router.openshift.io/https_header</code>	Sets a Strict-Transport-Security header for the edge terminated or re-encrypt route.	

**NOTE**

Environment variables can not be edited.

A route setting custom timeout

```

apiVersion: v1
kind: Route
metadata:
  annotations:
    haproxy.router.openshift.io/timeout: 5500ms 1
...

```

1

Specifies the new timeout with HAProxy supported units (**us**, **ms**, **s**, **m**, **h**, **d**). If the unit is not provided, **ms** is the default.

**NOTE**

Setting a server-side timeout value for passthrough routes too low can cause WebSocket connections to timeout frequently on that route.

10.2. SECURED ROUTES

The following sections describe how to create re-encrypt and edge routes with custom certificates.

**IMPORTANT**

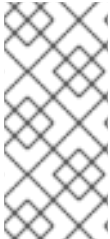
If you create routes in Microsoft Azure through public endpoints, the resource names are subject to restriction. You cannot create resources that use certain terms. For a list of terms that Azure restricts, see [Resolve reserved resource name errors](#) in the Azure documentation.

10.2.1. 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 reencrypt TLS termination. The following assumes that the certificate/key pair are in the **tls.crt** and **tls.key** files in the current working directory. You must also specify a destination CA certificate to enable the Ingress Controller to trust the service's certificate. You may also specify a CA certificate if needed to complete the certificate chain. Substitute the actual path names for **tls.crt**, **tls.key**, **cacert.crt**, and (optionally) **ca.crt**. Substitute the name of the **Service** resource that you want to expose for **frontend**. Substitute the appropriate host name for **www.example.com**.

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

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

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

YAML Definition of the Secure Route

```
apiVersion: v1
kind: Route
metadata:
  name: frontend
spec:
  host: www.example.com
  to:
    kind: Service
    name: frontend
  tls:
    termination: reencrypt
    key: |-
      -----BEGIN PRIVATE KEY-----
      [...]
      -----END PRIVATE KEY-----
    certificate: |-
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
```

```

caCertificate: |-
-----BEGIN CERTIFICATE-----
[...]
-----END CERTIFICATE-----
destinationCACertificate: |-
-----BEGIN CERTIFICATE-----
[...]
-----END CERTIFICATE-----

```

See **oc create route reencrypt --help** for more options.

10.2.2. 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:

```
$ 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.

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

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

YAML Definition of the Secure Route

```
apiVersion: v1
```

```
kind: Route
metadata:
  name: frontend
spec:
  host: www.example.com
  to:
    kind: Service
    name: frontend
  tls:
    termination: edge
    key: |-
      -----BEGIN PRIVATE KEY-----
      [...]
      -----END PRIVATE KEY-----
    certificate: |-
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
    caCertificate: |-
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
```

See **oc create route edge --help** for more options.

CHAPTER 11. CONFIGURING INGRESS CLUSTER TRAFFIC

11.1. 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 of preference:

- If you have HTTP/HTTPS, use an Ingress Controller.
- If you have a TLS-encrypted protocol other than HTTPS. For example, for TLS with the SNI header, use an Ingress Controller.
- Otherwise, use a Load Balancer, an External IP, or a **NodePort**.

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

11.2. 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.

11.2.1. 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.

11.2.2. Creating a project and service

If the project and service that you want to expose do not exist, first create the project, then the service.

If the project and service already exist, skip to the procedure on exposing the service to create a route.

Prerequisites

- Install the **oc** CLI and log in as a cluster administrator.

Procedure

1. Create a new project for your service:

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

For example:

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

2. Use the **oc new-app** command to create a service. For example:

```
$ oc new-app \
  -e MYSQL_USER=admin \
  -e MYSQL_PASSWORD=redhat \
  -e MYSQL_DATABASE=mysqlldb \
  registry.redhat.io/rhsc/mysql-80-rhel7
```

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

```
$ oc get svc -n myproject
NAME          TYPE        CLUSTER-IP   EXTERNAL-IP  PORT(S)    AGE
mysql-80-rhel7 ClusterIP    172.30.63.31 <none>       3306/TCP   4m55s
```

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

11.2.3. Exposing the service by creating a route

You can expose the service as a route by using the **oc expose** command.

Procedure

To expose the service:

1. Log in to OpenShift Container Platform.
2. Log in to the project where the service you want to expose is located:

```
$ oc project project1
```

3. Run the following command to expose the route:

```
$ oc expose service <service_name>
```

For example:

```
$ oc expose service mysql-80-rhel7
route "mysql-80-rhel7" exposed
```

4. Use a tool, such as cURL, to make sure you can reach the service using the cluster IP address for the service:

```
$ curl <pod_ip>:<port>
```

For example:

```
$ curl 172.30.131.89:3306
```

The examples in this section use a MySQL service, which requires a client application. If you get a string of characters with the **Got packets out of order** message, you are connected to the service.

If you have a MySQL client, log in with the standard CLI command:

```
■
```

```
$ mysql -h 172.30.131.89 -u admin -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.

MySQL [(none)]>
```

11.2.4. 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**.

11.2.5. 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:

```
# 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**.

11.2.6. 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](#).

11.3. 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.

11.3.1. 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.

11.3.2. Creating a project and service

If the project and service that you want to expose do not exist, first create the project, then the service.

If the project and service already exist, skip to the procedure on exposing the service to create a route.

Prerequisites

- Install the **oc** CLI and log in as a cluster administrator.

Procedure

1. Create a new project for your service:

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

For example:

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

2. Use the **oc new-app** command to create a service. For example:

```
$ oc new-app \  
-e MYSQL_USER=admin \  

```

```
-e MYSQL_PASSWORD=redhat \
-e MYSQL_DATABASE=mysqlpdb \
registry.redhat.io/rhsc1/mysql-80-rhel7
```

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

```
$ oc get svc -n myproject
NAME          TYPE        CLUSTER-IP   EXTERNAL-IP  PORT(S)    AGE
mysql-80-rhel7 ClusterIP    172.30.63.31 <none>       3306/TCP    4m55s
```

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

11.3.3. Exposing the service by creating a route

You can expose the service as a route by using the **oc expose** command.

Procedure

To expose the service:

1. Log in to OpenShift Container Platform.
2. Log in to the project where the service you want to expose is located:

```
$ oc project project1
```

3. Run the following command to expose the route:

```
$ oc expose service <service_name>
```

For example:

```
$ oc expose service mysql-80-rhel7
route "mysql-80-rhel7" exposed
```

4. Use a tool, such as cURL, to make sure you can reach the service using the cluster IP address for the service:

```
$ curl <pod_ip>:<port>
```

For example:

```
$ curl 172.30.131.89:3306
```

The examples in this section use a MySQL service, which requires a client application. If you get a string of characters with the **Got packets out of order** message, you are connected to the service.

If you have a MySQL client, log in with the standard CLI command:

```
$ mysql -h 172.30.131.89 -u admin -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
```

```
MySQL [(none)]>
```

11.3.4. Creating a load balancer service

Use the following procedure to create a load balancer service.

Prerequisites

- Make sure that the project and service you want to expose exist.

Procedure

To create a load balancer service:

1. Log in to OpenShift Container Platform.
2. Load the project where the service you want to expose is located.

```
$ oc project project1
```

3. Open a text file on the master node and paste the following text, editing the file as needed:

Sample load balancer configuration file

```
apiVersion: v1
kind: Service
metadata:
  name: egress-2 1
spec:
  ports:
    - name: db
      port: 3306 2
  loadBalancerIP:
  type: LoadBalancer 3
  selector:
    name: mysql 4
```

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

4. Save and exit the file.
5. Run the following command to create the service:

```
oc create -f <file-name>
```

For example:

```
oc create -f mysql-lb.yaml
```

6. Execute the following command to view the new service:

```
$ oc get svc
NAME      TYPE      CLUSTER-IP    EXTERNAL-IP      PORT(S)
AGE
egress-2  LoadBalancer  172.30.22.226  ad42f5d8b303045-487804948.example.com
3306:30357/TCP  15m
```

The service has an external IP address automatically assigned if there is a cloud provider enabled.

7. On the master, use a tool, such as cURL, to make sure you can reach the service using the public IP address:

```
$ curl <public-ip>:<port>
```

For example:

```
$ curl 172.29.121.74:3306
```

The examples in this section use a MySQL service, which requires a client application. If you get a string of characters with the **Got packets out of order** message, you are connecting with the service:

If you have a MySQL client, log in with the standard CLI command:

```
$ mysql -h 172.30.131.89 -u admin -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.

MySQL [(none)]>
```

11.4. 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.

11.4.1. 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.

11.4.2. Creating a project and service

If the project and service that you want to expose do not exist, first create the project, then the service.

If the project and service already exist, skip to the procedure on exposing the service to create a route.

Prerequisites

- Install the **oc** CLI and log in as a cluster administrator.

Procedure

1. Create a new project for your service:

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

For example:

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

2. Use the **oc new-app** command to create a service. For example:

```
$ oc new-app \  
-e MYSQL_USER=admin \  
-e MYSQL_PASSWORD=redhat \  

```

```
-e MYSQL_DATABASE=mysqldb \
registry.redhat.io/rhsc1/mysql-80-rhel7
```

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

```
$ oc get svc -n myproject
NAME          TYPE        CLUSTER-IP   EXTERNAL-IP  PORT(S)    AGE
mysql-80-rhel7 ClusterIP    172.30.63.31 <none>       3306/TCP   4m55s
```

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

11.4.3. Exposing the service by creating a route

You can expose the service as a route by using the **oc expose** command.

Procedure

To expose the service:

1. Log in to OpenShift Container Platform.
2. Log in to the project where the service you want to expose is located:

```
$ oc project project1
```

3. Run the following command to expose the route:

```
$ oc expose service <service_name>
```

For example:

```
$ oc expose service mysql-80-rhel7
route "mysql-80-rhel7" exposed
```

4. Use a tool, such as cURL, to make sure you can reach the service using the cluster IP address for the service:

```
$ curl <pod_ip>:<port>
```

For example:

```
$ curl 172.30.131.89:3306
```

The examples in this section use a MySQL service, which requires a client application. If you get a string of characters with the **Got packets out of order** message, you are connected to the service.

If you have a MySQL client, log in with the standard CLI command:

```
$ mysql -h 172.30.131.89 -u admin -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.

MySQL [(none)]>
```



11.5. 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**.

11.5.1. 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:

```
$ 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.

11.5.2. Creating a project and service

If the project and service that you want to expose do not exist, first create the project, then the service.

If the project and service already exist, skip to the procedure on exposing the service to create a route.

Prerequisites

- Install the **oc** CLI and log in as a cluster administrator.

Procedure

1. Create a new project for your service:

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

For example:

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

2. Use the **oc new-app** command to create a service. For example:

```
$ oc new-app \
  -e MYSQL_USER=admin \
  -e MYSQL_PASSWORD=redhat \
  -e MYSQL_DATABASE=mysqldb \
  registry.redhat.io/rhsc/mysql-80-rhel7
```

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

```
$ oc get svc -n myproject
NAME          TYPE        CLUSTER-IP   EXTERNAL-IP  PORT(S)    AGE
mysql-80-rhel7 ClusterIP    172.30.63.31 <none>       3306/TCP   4m55s
```

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

11.5.3. Exposing the service by creating a route

You can expose the service as a route by using the **oc expose** command.

Procedure

To expose the service:

1. Log in to OpenShift Container Platform.
2. Log in to the project where the service you want to expose is located:

```
$ oc project project1
```

3. To expose a node port for the application, enter the following command. OpenShift Container Platform automatically selects an available port in the **30000-32767** range.

```
$ oc expose dc mysql-80-rhel7 --type=NodePort --name=mysql-ingress
```

4. Optional: To confirm the service is available with a node port exposed, enter the following command:

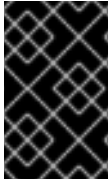
```
$ oc get svc -n myproject
NAME          TYPE        CLUSTER-IP   EXTERNAL-IP  PORT(S)    AGE
mysql-80-rhel7 ClusterIP    172.30.217.127 <none>       3306/TCP   9m44s
mysql-ingress  NodePort    172.30.107.72  <none>       3306:31345/TCP 39s
```

-
- 5. Optional: To remove the service created automatically by the **oc new-app** command, enter the following command:

```
$ oc delete svc mysql-80-rhel7
```

CHAPTER 12. CONFIGURING THE CLUSTER-WIDE PROXY

Production environments can deny direct access to the Internet and instead have an HTTP or HTTPS proxy available. You can configure OpenShift Container Platform to use a proxy by [modifying the Proxy object for existing clusters](#) or by configuring the proxy settings in the `install-config.yaml` file for new clusters.



IMPORTANT

The cluster-wide proxy is only supported if you used a user-provisioned infrastructure installation or provide your own networking, such as a virtual private cloud or virtual network, for a supported provider.

Prerequisites

- Review the [sites that your cluster requires access to](#)

and determine whether any of them must bypass the proxy. By default, all cluster egress traffic is proxied, including calls to the cloud provider API for the cloud that hosts your cluster. Add sites to the Proxy object's `spec.noProxy` field to bypass the proxy if necessary.



NOTE

The Proxy object's **status.noProxy** field is populated by default with the instance metadata endpoint (**169.254.169.254**) and with the values of the **networking.machineCIDR**, **networking.clusterNetwork.cidr**, and **networking.serviceNetwork** fields from your installation configuration.

12.1. 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:

```
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:

```
apiVersion: v1
data:
  ca-bundle.crt: | 1
    <MY_PEM_ENCODED_CERTS> 2
kind: ConfigMap
metadata:
  name: user-ca-bundle 3
  namespace: openshift-config 4
```

- 1** This data key must be named **ca-bundle.crt**.
- 2** One or more PEM-encoded X.509 certificates used to sign the proxy's identity certificate.
- 3** The 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:

```
$ oc create -f user-ca-bundle.yaml
```

2. Use the **oc edit** command to modify the Proxy object:

```
$ oc edit proxy/cluster
```

3. Configure the necessary fields for the proxy:

```
apiVersion: config.openshift.io/v1
kind: Proxy
metadata:
  name: cluster
spec:
  httpProxy: http://<username>:<pswd>@<ip>:<port> 1
  httpsProxy: http://<username>:<pswd>@<ip>:<port> 2
```

```
noProxy: example.com 3
readinessEndpoints:
- http://www.google.com 4
- https://www.google.com
trustedCA:
  name: user-ca-bundle 5
```

- 1 A proxy URL to use for creating HTTP connections outside the cluster. The URL scheme must be **http**.
- 2 A proxy URL to use for creating HTTPS connections outside the cluster. If this is not specified, then **httpProxy** is used for both HTTP and HTTPS connections. The URL scheme must be **http**; **https** is currently not supported.
- 3 A comma-separated list of destination domain names, domains, IP addresses or other network 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.

12.2. 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 13. 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.

13.1. 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:

```
apiVersion: v1
baseDomain: my.domain.com
proxy:
  httpProxy: http://<username>:<pswd>@<ip>:<port> 1
  httpsProxy: http://<username>:<pswd>@<ip>:<port> 2
  noProxy: example.com 3
additionalTrustBundle: | 4
  -----BEGIN CERTIFICATE-----
  <MY_TRUSTED_CA_CERT>
  -----END CERTIFICATE-----
...
```

- 1 A proxy URL to use for creating HTTP connections outside the cluster. The URL scheme must be **http**. If you use an MITM transparent proxy network that does not require additional proxy configuration but requires additional CAs, you must not specify an **httpProxy** value.
- 2 A proxy URL to use for creating HTTPS connections outside the cluster. If this field is not specified, then **httpProxy** is used for both HTTP and HTTPS connections. The URL scheme must be **http**; **https** is currently not supported. If you use an MITM transparent proxy network that does not require additional proxy configuration but requires additional CAs, you must not specify an **httpsProxy** value.
- 3 A comma-separated list of destination domain names, domains, IP addresses, or other network CIDRs to exclude proxying. Preface a domain with **.** to include all subdomains of that domain. Use ***** to bypass proxy for all destinations.
- 4 If provided, the installation program generates a ConfigMap that is named **user-ca-bundle** in the **openshift-config** namespace that contains one or more additional CA certificates that are required for proxying HTTPS connections. The Cluster Network Operator then creates a **trusted-ca-bundle** ConfigMap that merges these contents with the Red Hat Enterprise Linux CoreOS (RHCOS) trust bundle, and this ConfigMap is referenced in the Proxy object's **trustedCA** field. The **additionalTrustBundle** field is required unless the proxy's identity certificate is signed by an authority from the RHCOS trust bundle. If you use an MITM transparent proxy network that does not require additional proxy configuration but requires additional CAs, you must provide the MITM CA certificate.



NOTE

The installation program does not support the proxy **readinessEndpoints** field.

2. Save the file and reference it when installing OpenShift Container Platform.

The installation program creates a cluster-wide proxy that is named **cluster** that uses the proxy settings in the provided **install-config.yaml** file. If no proxy settings are provided, a **cluster** Proxy object is still created, but it will have a nil **spec**.

**NOTE**

Only the Proxy object named **cluster** is supported, and no additional proxies can be created.

13.2. ENABLING THE CLUSTER-WIDE PROXY

The Proxy object is used to manage the cluster-wide egress proxy. When a cluster is installed or upgraded without the proxy configured, a Proxy object is still generated but it will have a nil **spec**. For example:

```
apiVersion: config.openshift.io/v1
kind: Proxy
metadata:
  name: cluster
spec:
  trustedCA:
    name: ""
status:
```

A cluster administrator can configure the proxy for OpenShift Container Platform by modifying this **cluster** Proxy object.

**NOTE**

Only the Proxy object named **cluster** is supported, and no additional proxies can be created.

Prerequisites

- Cluster administrator permissions
- OpenShift Container Platform **oc** CLI tool installed

Procedure

1. Create a 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:

```
apiVersion: v1
data:
  ca-bundle.crt: | 1
    <MY_PEM_ENCODED_CERTS> 2
kind: ConfigMap
```

```

metadata:
  name: user-ca-bundle 3
  namespace: openshift-config 4

```

- 1 This data key must be named **ca-bundle.crt**.
- 2 One or more PEM-encoded X.509 certificates used to sign the proxy's identity certificate.
- 3 The 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:

```
$ oc create -f user-ca-bundle.yaml
```

2. Use the **oc edit** command to modify the Proxy object:

```
$ oc edit proxy/cluster
```

3. Configure the necessary fields for the proxy:

```

apiVersion: config.openshift.io/v1
kind: Proxy
metadata:
  name: cluster
spec:
  httpProxy: http://<username>:<pswd>@<ip>:<port> 1
  httpsProxy: http://<username>:<pswd>@<ip>:<port> 2
  noProxy: example.com 3
  readinessEndpoints:
    - http://www.google.com 4
    - https://www.google.com
  trustedCA:
    name: user-ca-bundle 5

```

- 1 A proxy URL to use for creating HTTP connections outside the cluster. The URL scheme must be **http**.
- 2 A proxy URL to use for creating HTTPS connections outside the cluster. If this is not specified, then **httpProxy** is used for both HTTP and HTTPS connections. The URL scheme must be **http**; **https** is currently not supported.
- 3 A comma-separated list of destination domain names, domains, IP addresses or other network 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

4. Save the file to apply the changes.

13.3. 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:

```
config.openshift.io/inject-trusted-cabundle="true"
```

The Operator mounts this ConfigMap into the container's local trust store.



NOTE

Adding a trusted CA certificate is only needed if the certificate is not included in the Red Hat Enterprise Linux CoreOS (RHCOS) trust bundle.

Certificate injection is not limited to Operators. The Cluster Network Operator injects certificates across any namespace when an empty ConfigMap is created with the **config.openshift.io/inject-trusted-cabundle=true** label.

The ConfigMap can reside in any namespace, but the ConfigMap must be mounted as a volume to each container within a Pod that requires a custom CA. For example:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-example-custom-ca-deployment
  namespace: my-example-custom-ca-ns
spec:
  ...
  spec:
    ...
    containers:
      - name: my-container-that-needs-custom-ca
        volumeMounts:
          - name: trusted-ca
            mountPath: /etc/pki/ca-trust/extracted/pem
            readOnly: true
        volumes:
          - name: trusted-ca
            configMap:
              name: trusted-ca
              items:
                - key: ca-bundle.crt 1
                  path: tls-ca-bundle.pem 2
```

- 1 **ca-bundle.crt** is required as the ConfigMap key.

- 2 **tls-ca-bundle.pem** is required as the ConfigMap path.