OpenShift Container Platform 4.5

Networking

Configuring and managing cluster networking

Last Updated: 2021-06-29

OpenShift Container Platform 4.5 Networking

Configuring and managing cluster networking

法律通告

摘要

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

目录

第 1 章 UNDERSTANDING NETWORKING 5
1.1. OPENSHIFT CONTAINER PLATFORM DNS 5

第 2 章 ACCESSING HOSTS 6
2.1. ACCESSING HOSTS ON AMAZON WEB SERVICES IN AN INSTALLER-PROVISIONED INFRASTRUCTURE CLUSTER 6

第 3 章 CLUSTER NETWORK OPERATOR IN OPENSHIFT CONTAINER PLATFORM 7
3.1. CLUSTER NETWORK OPERATOR 7
3.2. VIEWING THE CLUSTER NETWORK CONFIGURATION 7
3.3. VIEWING CLUSTER NETWORK OPERATOR STATUS 8
3.4. VIEWING CLUSTER NETWORK OPERATOR LOGS 8
3.5. CLUSTER NETWORK OPERATOR CONFIGURATION 9

第 4 章 DNS OPERATOR IN OPENSHIFT CONTAINER PLATFORM 12
4.1. DNS OPERATOR 12
4.2. VIEW THE DEFAULT DNS 12
4.3. USING DNS FORWARDING 13
4.4. DNS OPERATOR STATUS 15
4.5. DNS OPERATOR LOGS 15

第 5 章 INGRESS OPERATOR IN OPENSHIFT CONTAINER PLATFORM 16
5.1. THE INGRESS CONFIGURATION ASSET 16
5.2. INGRESS CONTROLLER CONFIGURATION PARAMETERS 16
5.3. VIEW THE DEFAULT INGRESS CONTROLLER 23
5.4. VIEW INGRESS OPERATOR STATUS 24
5.5. VIEW INGRESS CONTROLLER LOGS 24
5.6. VIEW INGRESS CONTROLLER STATUS 24

第 6 章 CONFIGURING THE INGRESS CONTROLLER 25
6.1. SETTING A CUSTOM DEFAULT CERTIFICATE 25
6.2. SCALING AN INGRESS CONTROLLER 26
6.3. CONFIGURING INGRESS ACCESS LOGGING 27
6.4. INGRESS CONTROLLER SHARDING 29
6.5. CONFIGURING AN INGRESS CONTROLLER TO USE AN INTERNAL LOAD BALANCER 31
6.6. CONFIGURING THE DEFAULT INGRESS CONTROLLER FOR YOUR CLUSTER TO BE INTERNAL 32
6.7. CONFIGURING THE ROUTE ADMISSION POLICY 33
6.8. USING WILDCARD ROUTES 34
6.9. ENABLING HTTP/2 INGRESS CONNECTIVITY 34
6.10. ADDITIONAL RESOURCES 35

第 7 章 USING THE STREAM CONTROL TRANSMISSION PROTOCOL (SCTP) ON A BARE METAL CLUSTER . 36
7.1. SUPPORT FOR STREAM CONTROL TRANSMISSION PROTOCOL (SCTP) ON OPENSHIFT CONTAINER PLATFORM 36
7.2. ENABLING STREAM CONTROL TRANSMISSION PROTOCOL (SCTP) 37
7.3. VERIFYING STREAM CONTROL TRANSMISSION PROTOCOL (SCTP) IS ENABLED 38

第 8 章 CONFIGURING PTP HARDWARE 41
8.1. ABOUT PTP HARDWARE 41
8.2. AUTOMATED DISCOVERY OF PTP NETWORK DEVICES 41
8.3. INSTALLING THE PTP OPERATOR 42
8.4. CONFIGURING LINUXPTP SERVICES 44

第 9 章 NETWORK POLICY 47
9.1. ABOUT NETWORK POLICY 47
9.2. CREATING A NETWORK POLICY 50
9.3. VIEWING A NETWORK POLICY 52
9.4. EDITING A NETWORK POLICY 54
9.5. DELETING A NETWORK POLICY 56
9.6. DEFINING A DEFAULT NETWORK POLICY FOR PROJECTS 57
9.7. CONFIGURING MULTITENANT ISOLATION WITH NETWORK POLICY 59

第 10 章 MULTIPLE NETWORKS 63
10.1. UNDERSTANDING MULTIPLE NETWORKS 63
10.2. ATTACHING A POD TO AN ADDITIONAL NETWORK 64
10.3. REMOVING A POD FROM AN ADDITIONAL NETWORK 69
10.4. CONFIGURING A BRIDGE NETWORK 70
10.5. CONFIGURING A HOST-DEVICE NETWORK 77
10.6. CONFIGURING AN IPVLAN NETWORK 83
10.7. CONFIGURING A MACVLAN NETWORK WITH BASIC CUSTOMIZATIONS 90
10.8. CONFIGURING A MACVLAN NETWORK 94
10.9. EDITING AN ADDITIONAL NETWORK 101
10.10. REMOVING AN ADDITIONAL NETWORK 101

第 11 章 HARDWARE NETWORKS 103
11.1. ABOUT SINGLE ROOT I/O VIRTUALIZATION (SR-IOV) HARDWARE NETWORKS 103
11.2. INSTALLING THE SR-IOV NETWORK OPERATOR 107
11.3. CONFIGURING THE SR-IOV NETWORK OPERATOR 110
11.4. CONFIGURING AN SR-IOV NETWORK DEVICE 113
11.5. CONFIGURING AN SR-IOV ETHERNET NETWORK ATTACHMENT 117
11.6. ADDING A POD TO AN SR-IOV ADDITIONAL NETWORK 123
11.7. USING HIGH PERFORMANCE MULTICAST 127
11.8. USING VIRTUAL FUNCTIONS (VFS) WITH DPDK AND RDMA MODES 129

第 12 章 OPENSHIFT SDN DEFAULT CNI NETWORK PROVIDER 139
12.1. ABOUT THE OPENSHIFT SDN DEFAULT CNI NETWORK PROVIDER 139
12.2. CONFIGURING EGRESS IPS FOR A PROJECT 140
12.3. CONFIGURING AN EGRESS FIREWALL TO CONTROL ACCESS TO EXTERNAL IP ADDRESSES 144
12.4. EDITING AN EGRESS FIREWALL FOR A PROJECT 148
12.5. REMOVING AN EGRESS FIREWALL FROM A PROJECT 150
12.6. CONSIDERATIONS FOR THE USE OF AN EGRESS ROUTER POD 150
12.7. DEPLOYING AN EGRESS ROUTER POD IN REDIRECT MODE 153
12.8. DEPLOYING AN EGRESS ROUTER POD IN HTTP PROXY MODE 156
12.9. DEPLOYING AN EGRESS ROUTER POD IN DNS PROXY MODE 158
12.10. CONFIGURING AN EGRESS ROUTER POD DESTINATION LIST FROM A CONFIG MAP 161
12.11. ENABLING MULTICAST FOR A PROJECT 163
12.12. DISABLING MULTICAST FOR A PROJECT 165
12.13. CONFIGURING NETWORK ISOLATION USING OPENSHIFT SDN 165
12.14. CONFIGURING KUBE-PROXY 167

第 13 章 OVN-KUBERNETES DEFAULT CNI NETWORK PROVIDER 170
13.1. ABOUT THE OVN-KUBERNETES DEFAULT CONTAINER NETWORK INTERFACE (CNI) NETWORK PROVIDER 170
13.2. MIGRATE FROM THE OPENSHIFT SDN DEFAULT CNI NETWORK PROVIDER 171
13.3. ROLLBACK TO THE OPENSHIFT SDN NETWORK PROVIDER 173
13.4. ENABLING MULTICAST FOR A PROJECT 175
13.5. DISABLING MULTICAST FOR A PROJECT 177

第 14 章 CONFIGURING ROUTES 179
14.1. ROUTE CONFIGURATION 179
14.2. SECURED ROUTES 186

第 15 章 CONFIGURING INGRESS CLUSTER TRAFFIC 190
15.1. CONFIGURING INGRESS CLUSTER TRAFFIC OVERVIEW 190
15.2. CONFIGURING EXTERNALIPS FOR SERVICES 190
15.3. CONFIGURING INGRESS CLUSTER TRAFFIC USING AN INGRESS CONTROLLER 196
15.4. CONFIGURING INGRESS CLUSTER TRAFFIC USING A LOAD BALANCER 201
15.5. CONFIGURING INGRESS CLUSTER TRAFFIC FOR A SERVICE EXTERNAL IP 205
15.6. CONFIGURING INGRESS CLUSTER TRAFFIC USING A NODEPORT 206

第 16 章 CONFIGURING THE CLUSTER-WIDE PROXY 210
16.1. PREREQUISITES 210
16.2. ENABLING THE CLUSTER-WIDE PROXY 210
16.3. REMOVING THE CLUSTER-WIDE PROXY 212

第 17 章 CONFIGURING A CUSTOM PKI 214
17.1. CONFIGURING THE CLUSTER-WIDE PROXY DURING INSTALLATION 214
17.2. ENABLING THE CLUSTER-WIDE PROXY 216
17.3. CERTIFICATE INJECTION USING OPERATORS 218

第 18 章 LOAD BALANCING ON RHOSP 220
18.1. USING THE OCTAVIA OVN LOAD BALANCER PROVIDER DRIVER WITH KURYR SDN 220
18.2. SCALING CLUSTERS FOR APPLICATION TRAFFIC BY USING OCTAVIA 221
18.3. SCALING FOR INGRESS TRAFFIC BY USING RHOSP OCTAVIA 223

第1章 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.

注意

Some cloud platforms offer metadata APIs that listen on the 169.254.169.254 IP address, a link-local IP address in the IPv4 **169.254.0.0/16** CIDR block.

This CIDR block is not reachable from the pod network. Pods that need access to these IP addresses must be given host network access by setting the **spec.hostNetwork** field in the pod spec to **true**.

If you allow a pod host network access, you grant the pod privileged access to the underlying network infrastructure.

1.1. OPENSHIFT CONTAINER PLATFORM DNS

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

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

第 2 章 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 (RHCOS), 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.

   注意

   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>
```

第 3 章 CLUSTER NETWORK OPERATOR IN OPENSHIFT CONTAINER PLATFORM

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

3.1. CLUSTER NETWORK OPERATOR

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

Procedure

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

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

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

Example output

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

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

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

Example output

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

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

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
```

Example output

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

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

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

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
```

重要

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: 1
  - cidr: 10.128.0.0/14
    hostPrefix: 23
  serviceNetwork: 2
  - 172.30.0.0/16
  defaultNetwork: 3
    ...
  kubeProxyConfig: 4
    iptablesSyncPeriod: 30s 5
    proxyArguments:
      iptables-min-sync-period: 6
      - 0s
```

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

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

3. Configures the default CNI network provider for the cluster network.

4. The parameters for this object specify the Kubernetes network proxy (kube-proxy) configuration. If you are using the OVN-Kubernetes default CNI 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.

注意

Because of performance improvements introduced in OpenShift Container Platform 4.3 and greater, adjusting the **iptablesSyncPeriod** parameter is no longer necessary.

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 default CNI network provider

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

注意

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

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

1 The default CNI network provider plug-in that is used.

2 OpenShift SDN specific configuration parameters.

3 The network isolation mode for OpenShift SDN.

4 The maximum transmission unit (MTU) for the VXLAN overlay network. This value is normally configured automatically.

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

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

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

注意

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

```
defaultNetwork:
  type: OVNKubernetes 1
  ovnKubernetesConfig: 2
    mtu: 1400 3
    genevePort: 6081 4
```

1 The default CNI network provider plug-in that is used.

2 OVN-Kubernetes specific configuration parameters.

3 The MTU for the Geneve (Generic Network Virtualization Encapsulation) 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:
      - 0s
```

第4章 DNS OPERATOR IN OPENSHIFT CONTAINER PLATFORM

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

4.1. DNS OPERATOR

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

Procedure

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

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

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

Example output

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

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

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

Example output

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

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

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
```

Example output

```
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}'
```

Example output

```
[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. If the forwarded zone is the Ingress domain managed by OpenShift Container Platform, then the upstream name server must be authorized for the domain.

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**.

注意

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 daemon set.

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 -n openshift-dns-operator deployment/dns-operator -c dns-operator
```

第 5 章 INGRESS OPERATOR IN OPENSHIFT CONTAINER PLATFORM

The Ingress Operator implements the **ingresscontroller** API and is the component responsible for enabling external access to OpenShift Container Platform cluster services. The Operator makes this possible by deploying and managing one or more HAProxy-based Ingress Controllers to handle routing. You can use the Ingress Operator to route traffic by specifying OpenShift Container Platform **Route** and Kubernetes **Ingress** resources.

5.1. THE INGRESS CONFIGURATION ASSET

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

YAML Definition of the Ingress resource

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

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

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

5.2. INGRESS CONTROLLER CONFIGURATION PARAMETERS

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

Parameter	Description

Parameter	Description
domain	**domain** is a DNS name serviced by the Ingress controller and is used to configure multiple features: • For the **LoadBalancerService** endpoint publishing strategy, **domain** is used to configure DNS records. See **endpointPublishingStrategy**. • When using a generated default certificate, the certificate is valid for **domain** and its **subdomains**. See **defaultCertificate**. • The value is published to individual Route statuses so that users know where to target external DNS records. The **domain** value must be unique among all Ingress controllers and cannot be updated. If empty, the default value is **ingress.config.openshift.io/cluster .spec.domain**.
replicas	**replicas** is the desired number of Ingress controller replicas. If not set, the default value is **2**.
endpointPublishingStrategy	**endpointPublishingStrategy** is used to publish the Ingress controller endpoints to other networks, enable load balancer integrations, and provide access to other systems. If not set, the default value is based on **infrastructure.config.openshift.io/cluster .status.platform**: • AWS: **LoadBalancerService** (with external scope) • Azure: **LoadBalancerService** (with external scope) • GCP: **LoadBalancerService** (with external scope) • Bare metal: **NodePortService** • Other: **HostNetwork** The **endpointPublishingStrategy** value cannot be updated.
defaultCertificate	The **defaultCertificate** value is a reference to a secret that contains the default certificate that is served by the Ingress controller. When Routes do not specify their own certificate, **defaultCertificate** is used. The secret must contain the following keys and data: * **tls.crt**: certificate file contents * **tls.key**: key file contents If not set, a wildcard certificate is automatically generated and used. The certificate is valid for the Ingress controller **domain** and **subdomains**, and the generated certificate's CA is automatically integrated with the cluster's trust store. The in-use certificate, whether generated or user-specified, is automatically integrated with OpenShift Container Platform built-in OAuth server.

Parameter	Description
namespaceSelector	**namespaceSelector** is used to filter the set of namespaces serviced by the Ingress controller. This is useful for implementing shards.
routeSelector	**routeSelector** is used to filter the set of Routes serviced by the Ingress controller. This is useful for implementing shards.

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

Parameter	Description
tlsSecurityProfile	**tlsSecurityProfile** specifies settings for TLS connections for Ingress controllers. If not set, the default value is based on the **apiservers.config.openshift.io/cluster** resource. When using the **Old**, **Intermediate**, and **Modern** profile types, the effective profile configuration is subject to change between releases. For example, given a specification to use the **Intermediate** profile deployed on release **X.Y.Z**, an upgrade to release **X.Y.Z+1** may cause a new profile configuration to be applied to the Ingress controller, resulting in a rollout. The minimum TLS version for Ingress controllers is **1.1**, and the maximum TLS version is **1.2**. **重要** 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**. 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**. **注意** Ciphers and the minimum TLS version of the configured security profile are reflected in the **TLSProfile** status.

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

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

注意

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.

警告

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

NodePortService endpoint publishing strategy

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

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

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

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

HostNetwork endpoint publishing strategy

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

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

5.3. VIEW THE DEFAULT INGRESS CONTROLLER

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

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

Procedure

- View the default Ingress Controller:

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

5.4. VIEW INGRESS OPERATOR STATUS

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

Procedure

- View your Ingress Operator status:

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

5.5. VIEW INGRESS CONTROLLER LOGS

You can view your Ingress Controller logs.

Procedure

- View your Ingress Controller logs:

```
$ oc logs --namespace=openshift-ingress-operator deployments/ingress-operator
```

5.6. VIEW INGRESS CONTROLLER STATUS

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>
```

第 6 章 CONFIGURING THE INGRESS CONTROLLER

6.1. SETTING A CUSTOM DEFAULT CERTIFICATE

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

Prerequisites

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

  ### Example output

  ```
  NAME      AGE
  default   10m
  ```

注意

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.

注意

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}'
```

Example output

```
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.

6.2. 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}'
```

Example output

```
2
```

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

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

Example output

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

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

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

Example output

```
3
```

注意

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

6.3. CONFIGURING INGRESS ACCESS LOGGING

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

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

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

Prerequisites

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

Procedure

Configure Ingress access logging to a sidecar.

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

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  replicas: 2
  endpointPublishingStrategy:
    type: NodePortService 1
  logging:
    access:
      destination:
        type: Container
```

1 **NodePortService** is not required to configure Ingress access logging to a sidecar. Ingress logging is compatible with any **endpointPublishingStrategy**.

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

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

Example output

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

Configure Ingress access logging to a Syslog endpoint.

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

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

注意

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

Configure Ingress access logging with a specific log format.

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

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

Disable Ingress access logging.

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

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

6.4. INGRESS CONTROLLER SHARDING

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

- Balance Ingress Controllers, or routers, with several routes to speed up responses to changes.
- Allocate certain routes to have different reliability guarantees than other routes.
- Allow certain Ingress Controllers to have different policies defined.
- Allow only specific routes to use additional features.
- Expose different routes on different addresses so that internal and external users can see different routes, for example.

Ingress Controller can use either route labels or namespace labels as a sharding method.

6.4.1. 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**.

6.4.2. 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
```

Example output

```
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**.

6.5. 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.

警告

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.

重要

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 CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

Procedure

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

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  namespace: openshift-ingress-operator
  name: <name> 1
spec:
  domain: <domain> 2
  endpointPublishingStrategy:
    type: LoadBalancerService
    loadBalancer:
      scope: Internal 3
```

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

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

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

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

```
$ oc create -f <name>-ingress-controller.yaml 1
```

1 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
```

6.6. 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.

> **警告**
>
> 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.

重要

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 CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

Procedure

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

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

6.7. CONFIGURING THE ROUTE ADMISSION POLICY

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

警告

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

Prerequisites

- Cluster administrator privileges.

Procedure

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

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

  Sample Ingress Controller configuration

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

6.8. USING WILDCARD ROUTES

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

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

Procedure

1. Configure the wildcard policy.

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

      ```
      $ oc edit IngressController
      ```

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

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

6.9. ENABLING HTTP/2 INGRESS CONNECTIVITY

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

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

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

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

重要

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

Procedure

Enable HTTP/2 on a single Ingress Controller.

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

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

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

Enable HTTP/2 on the entire cluster.

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

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

6.10. ADDITIONAL RESOURCES

- Configuring a custom PKI

第 7 章 USING THE STREAM CONTROL TRANSMISSION PROTOCOL (SCTP) ON A BARE METAL CLUSTER

As a cluster administrator, you can use the Stream Control Transmission Protocol (SCTP) on a cluster.

7.1. SUPPORT FOR STREAM CONTROL TRANSMISSION PROTOCOL (SCTP) ON OPENSHIFT CONTAINER PLATFORM

As a cluster administrator, you can enable SCTP on the hosts in the cluster. On Red Hat Enterprise Linux CoreOS (RHCOS), the SCTP module is disabled by default.

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

When enabled, you can use SCTP as a protocol with pods, services, and network policy. A **Service** object must be defined with the **type** parameter set to either the **ClusterIP** or **NodePort** value.

7.1.1. Example configurations using SCTP protocol

You can configure a pod or service to use SCTP by setting the **protocol** parameter to the **SCTP** value in the pod or service object.

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

```
apiVersion: v1
kind: Pod
metadata:
  namespace: project1
  name: example-pod
spec:
  containers:
    - name: example-pod
...
      ports:
        - containerPort: 30100
          name: sctpserver
          protocol: SCTP
```

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

```
apiVersion: v1
kind: Service
metadata:
  namespace: project1
  name: sctpserver
spec:
...
  ports:
    - name: sctpserver
      protocol: SCTP
      port: 30100
      targetPort: 30100
  type: ClusterIP
```

In the following example, a **NetworkPolicy** object is configured to apply to SCTP network traffic on port **80** from any pods with a specific label:

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

7.2. ENABLING STREAM CONTROL TRANSMISSION PROTOCOL (SCTP)

As a cluster administrator, you can load and enable the blacklisted SCTP kernel module on worker nodes in your cluster.

Prerequisites

- Install the OpenShift CLI (**oc**).
- Access to the cluster as a user with the **cluster-admin** role.

Procedure

1. Create a file named **load-sctp-module.yaml** that contains the following YAML definition:

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

2. To create the **MachineConfig** object, enter the following command:

```
$ oc create -f load-sctp-module.yaml
```

3. Optional: To watch the status of the nodes while the MachineConfig Operator applies the configuration change, enter the following command. When the status of a node transitions to **Ready**, the configuration update is applied.

```
$ oc get nodes
```

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

You can verify that SCTP is working on a cluster by creating a pod with an application that listens for SCTP traffic, associating it with a service, and then connecting to the exposed service.

Prerequisites

- Access to the Internet from the cluster to install the **nc** package.
- Install the OpenShift CLI (**oc**).
- Access to the cluster as a user with the **cluster-admin** role.

Procedure

1. Create a pod starts an SCTP listener:

   a. Create a file named **sctp-server.yaml** that defines a pod with the following YAML:

```
apiVersion: v1
kind: Pod
metadata:
  name: sctpserver
  labels:
    app: sctpserver
spec:
  containers:
    - name: sctpserver
      image: registry.access.redhat.com/ubi8/ubi
      command: ["/bin/sh", "-c"]
      args:
        ["dnf install -y nc && sleep inf"]
      ports:
        - containerPort: 30102
          name: sctpserver
          protocol: SCTP
```

   b. Create the pod by entering the following command:

```
$ oc create -f sctp-server.yaml
```

2. Create a service for the SCTP listener pod.

   a. Create a file named **sctp-service.yaml** that defines a service with the following YAML:

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

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

   ```
   $ oc create -f sctp-service.yaml
   ```

3. Create a pod for the SCTP client.

   a. Create a file named **sctp-client.yaml** with the following YAML:

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

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

   ```
   $ oc apply -f sctp-client.yaml
   ```

4. Run an SCTP listener on the server.

   a. To connect to the server pod, enter the following command:

   ```
   $ oc rsh sctpserver
   ```

b. To start the SCTP listener, enter the following command:

```
$ nc -l 30102 --sctp
```

5. Connect to the SCTP listener on the server.

a. Open a new terminal window or tab in your terminal program.

b. Obtain the IP address of the **sctpservice** service. Enter the following command:

```
$ oc get services sctpservice -o go-template='{{.spec.clusterIP}}{{"\n"}}'
```

c. To connect to the client pod, enter the following command:

```
$ oc rsh sctpclient
```

d. To start the SCTP client, enter the following command. Replace **<cluster_IP>** with the cluster IP address of the **sctpservice** service.

```
# nc <cluster_IP> 30102 --sctp
```

第 8 章 CONFIGURING PTP HARDWARE

重要

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

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

8.1. ABOUT PTP HARDWARE

OpenShift Container Platform includes the capability to use Precision Time Protocol (PTP)hardware on your nodes. You can configure linuxptp services on nodes in your cluster that have PTP-capable hardware.

注意

The PTP Operator works with PTP-capable devices on clusters provisioned only on bare metal infrastructure.

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 the following features:

- Discovery of the PTP-capable devices in a cluster.
- Management of the configuration of linuxptp services.

8.2. 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) object 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.

8.3. INSTALLING THE PTP OPERATOR

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

8.3.1. CLI: Installing the PTP Operator

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

Prerequisites

- A cluster installed on bare-metal hardware with nodes that have hardware that supports PTP.
- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

Procedure

1. To create a namespace for the PTP Operator, enter the following command:

```
$ cat << EOF| oc create -f -
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-ptp
  labels:
    name: openshift-ptp
    openshift.io/cluster-monitoring: "true"
EOF
```

2. To create an Operator group for the Operator, enter the following command:

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

3. Subscribe to the PTP Operator.

   a. Run the following command to set the OpenShift Container Platform major and minor version as an environment variable, which is used as the **channel** value in the next step.

   ```
   $ OC_VERSION=$(oc version -o yaml | grep openshiftVersion | \
       grep -o '[0-9]*[.][0-9]*' | head -1)
   ```

   b. To create a subscription for the PTP Operator, enter the following command:

   ```
   $ cat << EOF| oc create -f -
   apiVersion: operators.coreos.com/v1alpha1
   kind: Subscription
   metadata:
     name: ptp-operator-subscription
     namespace: openshift-ptp
   spec:
     channel: "${OC_VERSION}"
     name: ptp-operator
     source: redhat-operators
     sourceNamespace: openshift-marketplace
   EOF
   ```

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

   ```
   $ oc get csv -n openshift-ptp \
     -o custom-columns=Name:.metadata.name,Phase:.status.phase
   ```

   Example output

   ```
   Name                                         Phase
   ptp-operator.4.4.0-202006160135              Succeeded
   ```

8.3.2. Web console: Installing the PTP Operator

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

注意

You have to create the namespace and operator group as mentioned in the previous section.

Procedure

1. Install the PTP Operator using the OpenShift Container Platform web console:

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

    b. Choose **PTP Operator** from the list of available Operators, and then click **Install**.

    c. On the **Install Operator** page, under **A specific namespace on the cluster**select **openshift-ptp**. Then, click **Install**.

2. Optional: Verify that the PTP Operator installed successfully:

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

    b. Ensure that **PTP Operator** is listed in the **openshift-ptp** project with a **Status** of **InstallSucceeded**.

    注意

    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.

8.4. 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) object.

Prerequisites

- Install the OpenShift CLI (**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
```

Example output

```
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
I1115 09:41:17.117596 4143292 daemon.go:107] in applyNodePTPProfile
I1115 09:41:17.117604 4143292 daemon.go:109] updating NodePTPProfile to:
I1115 09:41:17.117607 4143292 daemon.go:110] ------------------------------------
I1115 09:41:17.117612 4143292 daemon.go:102] Profile Name: profile1 1
I1115 09:41:17.117616 4143292 daemon.go:102] Interface: ens787f1  2
I1115 09:41:17.117620 4143292 daemon.go:102] Ptp4lOpts: -s -2  3
I1115 09:41:17.117623 4143292 daemon.go:102] Phc2sysOpts: -a -r  4
I1115 09:41:17.117626 4143292 daemon.go:116] ------------------------------------
I1115 09:41:18.117934 4143292 daemon.go:186] Starting phc2sys...
I1115 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>}
I1115 09:41:19.118175 4143292 daemon.go:186] Starting ptp4l...
I1115 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.

第 9 章 NETWORK POLICY

9.1. ABOUT NETWORK POLICY

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

9.1.1. About network policy

In a cluster using a Kubernetes Container Network Interface (CNI) plug-in that supports Kubernetes network policy, network isolation is controlled entirely by **NetworkPolicy** objects. In OpenShift Container Platform 4.5, OpenShift SDN supports using network policy in its default network isolation mode.

注意

When using the OpenShift SDN cluster network provider, the following limitations apply regarding network policies:

- Egress network policy as specified by the **egress** field is not supported.
- IPBlock is supported by network policy, but without support for **except** clauses. If you create a policy with an IPBlock section that includes an **except** clause, the SDN pods log warnings and the entire IPBlock section of that policy is ignored.

警告

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

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

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

The following example **NetworkPolicy** objects demonstrate supporting different scenarios:

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

```
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.

  重要

  For the OVN-Kubernetes network provider plug-in, when the Ingress Controller is configured to use the **HostNetwork** endpoint publishing strategy, there is no supported way to apply network policy so that ingress traffic is allowed and all other traffic is denied.

  ```
  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. With OpenShift SDN, the **default** namespace is assigned the **netid:0** VNID and 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.

9.1.2. Optimizations for network policy

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

注意

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

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

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

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

- Reduce the number of OVS flow rules by using namespaces to contain groups of pods that need to be isolated.
  NetworkPolicy objects that select a whole namespace, by using the **namespaceSelector** or an empty **podSelector**, generate only a single OVS flow rule that matches the VXLAN virtual network ID (VNID) of the namespace.
- Keep the pods that do not need to be isolated in their original namespace, and move the pods that require isolation into one or more different namespaces.
- Create additional targeted cross-namespace network policies to allow the specific traffic that you do want to allow from the isolated pods.

9.1.3. Next steps

- Creating a network policy
- Optional: Defining a default network policy

9.1.4. Additional resources

- Projects and namespaces
- Configuring multitenant network policy
- NetworkPolicy API

9.2. CREATING A NETWORK POLICY

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

9.2.1. Creating a network policy

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

注意

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

Prerequisites

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

Procedure

1. Create a policy rule:

   a. Create a **<policy_name>.yaml** file:

   ```
   $ touch <policy_name>.yaml
   ```

   where:

   <policy_name>
   Specifies the network policy file name.

   b. Define a network policy in the file that you just created, such as in the following examples:

   #### Deny ingress from all pods in all namespaces

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

   #### Allow ingress from all pods in the same namespace

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

2. To create the network policy object, enter the following command:

   ```
   $ oc apply -f <policy_name>.yaml -n <namespace>
   ```

   where:

<policy_name>

Specifies the network policy file name.

<namespace>

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

Example output

```
networkpolicy "default-deny" created
```

9.2.2. Example NetworkPolicy object

The following annotates an example NetworkPolicy object:

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-27107 ①
spec:
  podSelector: ②
    matchLabels:
      app: mongodb
  ingress:
  - from:
    - podSelector: ③
        matchLabels:
          app: app
    ports: ④
    - protocol: TCP
      port: 27017
```

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

② A selector 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.

9.3. VIEWING A NETWORK POLICY

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

9.3.1. Viewing network policies

You can examine the network policies in a namespace.

注意

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

Prerequisites

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

Procedure

- List network policies in a namespace:
  - To view **NetworkPolicy** objects defined in a namespace, enter the following command:

    ```
    $ oc get networkpolicy
    ```

  - Optional: To examine a specific network policy, enter the following command:

    ```
    $ oc describe networkpolicy <policy_name> -n <namespace>
    ```

    where:

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

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

    For example:

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

    #### Output for **oc describe** command

    ```
    Name:         allow-same-namespace
    Namespace:    ns1
    Created on:   2021-05-24 22:28:56 -0400 EDT
    Labels:       <none>
    Annotations:  <none>
    Spec:
      PodSelector:     <none> (Allowing the specific traffic to all pods in this namespace)
      Allowing ingress traffic:
        To Port: <any> (traffic allowed to all ports)
        From:
          PodSelector: <none>
      Not affecting egress traffic
      Policy Types: Ingress
    ```

9.3.2. Example NetworkPolicy object

The following annotates an example NetworkPolicy object:

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-27107 1
spec:
  podSelector: 2
    matchLabels:
      app: mongodb
  ingress:
  - from:
    - podSelector: 3
        matchLabels:
          app: app
    ports: 4
    - protocol: TCP
      port: 27017
```

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

2 A selector 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.

9.4. EDITING A NETWORK POLICY

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

9.4.1. Editing a network policy

You can edit a network policy in a namespace.

注意

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

Prerequisites

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

- You are working in the namespace where the network policy exists.

Procedure

1. Optional: To list the network policy objects in a namespace, enter the following command:

   ```
   $ oc get networkpolicy -n <namespace>
   ```

   where:

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

2. Edit the **NetworkPolicy** object.

   - If you saved the network policy definition in a file, edit the file and make any necessary changes, and then enter the following command.

     ```
     $ oc apply -n <namespace> -f <policy_file>.yaml
     ```

     where:

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

     <policy_file>
     Specifies the name of the file containing the network policy.

   - If you need to update the **NetworkPolicy** object directly, enter the following command:

     ```
     $ oc edit networkpolicy <policy_name> -n <namespace>
     ```

     where:

     <policy_name>
     Specifies the name of the network policy.

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

3. Confirm that the **NetworkPolicy** object is updated.

   ```
   $ oc describe networkpolicy <policy_name> -n <namespace>
   ```

   where:

   <policy_name>
   Specifies the name of the network policy.

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

9.4.2. Example NetworkPolicy object

The following annotates an example NetworkPolicy object:

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-27107 1
spec:
  podSelector: 2
    matchLabels:
      app: mongodb
  ingress:
  - from:
    - podSelector: 3
        matchLabels:
          app: app
    ports: 4
    - protocol: TCP
      port: 27017
```

1. The **name** of the NetworkPolicy object.
2. A selector 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.

9.4.3. Additional resources

- Creating a network policy

9.5. DELETING A NETWORK POLICY

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

9.5.1. Deleting a network policy

You can delete a network policy in a namespace.

注意

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

Prerequisites

- Your cluster uses a cluster network provider that supports **NetworkPolicy** objects, such as the OpenShift SDN network provider with **mode: NetworkPolicy** set. This mode is the default for OpenShift SDN.

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

Procedure

- To delete a **NetworkPolicy** object, enter the following command:

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

  where:

  <policy_name>
  Specifies the name of the network policy.

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

  Example output

  ```
  networkpolicy.networking.k8s.io/allow-same-namespace deleted
  ```

9.6. DEFINING A DEFAULT NETWORK POLICY FOR PROJECTS

As a cluster administrator, you can modify the new project template to automatically include network policies when you create a new project. If you do not yet have a customized template for new projects, you must first create one.

9.6.1. Modifying the template for new projects

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

To create your own custom project template:

Procedure

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

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

3. Use a text editor to modify the generated **template.yaml** file by adding objects or modifying existing objects.
4. The project template must be created in the **openshift-config** namespace. Load your modified template:

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

5. Edit the project configuration resource using the web console or CLI.
   - Using the web console:
     i. Navigate to the **Administration** → **Cluster Settings** page.
     ii. Click **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.

9.6.2. Adding network policies to the new project template

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

Prerequisites

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

9.7. CONFIGURING MULTITENANT ISOLATION WITH NETWORK POLICY

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

注意

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

9.7.1. Configuring multitenant isolation by using network policy

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

Prerequisites

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

Procedure

1. Create the following **NetworkPolicy** objects:

   a. A policy named **allow-from-openshift-ingress**.

   重要

   For the OVN-Kubernetes network provider plug-in, when the Ingress Controller is configured to use the **HostNetwork** endpoint publishing strategy, there is no supported way to apply network policy so that ingress traffic is allowed and all other traffic is denied.

```
$ cat << EOF| oc create -f -
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
EOF
```

   b. A policy named **allow-from-openshift-monitoring**:

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

c. A policy named **allow-same-namespace**:

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

2. 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'
   ```

3. 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 -n project1 networkpolicy allow-from-openshift-ingress -o yaml
   ```

Example output

```
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
```

9.7.2. Next steps

- Defining a default network policy

9.7.3. Additional resources

- OpenShift SDN network isolation modes

第 10 章 MULTIPLE NETWORKS

10.1. 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.

10.1.1. 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 **NetworkAttachmentDefinition** custom resource (CR). A CNI configuration inside each of these CRs defines how that interface is created.

10.1.2. 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**: Configuring a host-device additional network allows pods access to a physical Ethernet network device on the host system.

- **ipvlan**: Configuring 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.

- **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.
  A macvlan additional network can be configured in two ways:
  - Configuring a macvlan-based additional network with basic customizations
  - Configuring a macvlan-based additional network
- **SR-IOV**: Configuring 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.

10.2. ATTACHING A POD TO AN ADDITIONAL NETWORK

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

10.2.1. 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.

When a pod is created additional networks are attached to it. However, if a pod already exists, you cannot attach additional networks to it.

The pod must be in the same namespace as the additional network.

Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in to the cluster.

Procedure

1. Add an annotation to the **Pod** object. Only one of the following annotation formats can be used:

   a. To attach an additional network without any customization, add an annotation with the following format. Replace **<network>** with the name of the additional network to associate with the pod:

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

   1 To specify more than one additional network, separate each network with a comma. Do not include whitespace between the comma. If you specify the same additional network multiple times, that pod will have multiple network interfaces attached to that network.

   b. To attach an additional network with customizations, add an annotation with the following format:

   ```
   metadata:
     annotations:
   ```

```
k8s.v1.cni.cncf.io/networks: |-
  [
    {
      "name": "<network>", 1
      "namespace": "<namespace>", 2
      "default-route": ["<default-route>"] 3
    }
  ]
```

1. Specify the name of the additional network defined by a **NetworkAttachmentDefinition** object.

2. Specify the namespace where the **NetworkAttachmentDefinition** object is defined.

3. Optional: Specify an override for the default route, such as **192.168.17.1**.

2. To create the pod, enter the following command. Replace **<name>** with the name of the pod.

```
$ oc create -f <name>.yaml
```

3. Optional: To Confirm that the annotation exists in the **Pod** CR, enter the following command, replacing **<name>** with the name of the pod.

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

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

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

10.2.1.1. 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** resource, run the following command to edit its definition in the default editor. Replace **<name>** with the name of the **Pod** resource to edit.

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

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

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

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

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

   ```
   apiVersion: v1
   kind: Pod
   metadata:
     name: example-pod
   ```

```
  annotations:
    k8s.v1.cni.cncf.io/networks: '
    {
      "name": "net1"
    },
    {
      "name": "net2", ❶
      "default-route": ["192.0.2.1"] ❷
    }'
spec:
  containers:
  - name: example-pod
    command: ["/bin/bash", "-c", "sleep 2000000000000"]
    image: centos/tools
```

❶ The **name** key is the name of the additional network to associate with the pod.

❷ 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.

重要

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
```

注意

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

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

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

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

The following YAML describes the configuration parameters for the CNO:

Cluster Network Operator YAML configuration

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

1. Specify a name for the additional network attachment that you are creating. The name must be unique within the specified **namespace**.

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

3. Specify the CNI 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",
  "name": "<name>", 1
  "plugins": [{ 2
      "type": "macvlan",
      "capabilities": { "ips": true }, 3
      "master": "eth0", 4
      "mode": "bridge",
      "ipam": {
        "type": "static"
      }
    }, {
      "capabilities": { "mac": true }, 5
      "type": "tuning"
    }]
}
```

1. Specifies the name for the additional network attachment to create. The name must be unique within the specified **namespace**.

2. Specifies an array of CNI plug-in configurations. The first object specifies a macvlan plug-in configuration and the second object specifies a tuning plug-in configuration.

3. Specifies that a request is made to enable the static IP address functionality of the CNI plug-in runtime configuration capabilities.

4. Specifies the interface that the macvlan plug-in uses.

5. Specifies that a request is made to enable the static MAC address functionality of a CNI plug-in.

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

Edit the pod with:

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

macvlan CNI 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>", 1
        "ips": [ "192.0.2.205/24" ], 2
        "mac": "CA:FE:C0:FF:EE:00" 3
      }
    ]'
```

1 Use the **<name>** as provided when creating the **rawCNIConfig** above.

2 Provide an IP address including the subnet mask.

3 Provide the MAC address.

注意

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
```

10.3. REMOVING A POD FROM AN ADDITIONAL NETWORK

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

10.3.1. Removing a pod from an additional network

You can remove a pod from an additional network only by deleting the pod.

Prerequisites

- An additional network is attached to the pod.
- Install the OpenShift CLI (**oc**).
- Log in to the cluster.

Procedure

- To delete the pod, enter the following command:

  ```
  $ oc delete pod <name> -n <namespace>
  ```

  - **<name>** is the name of the pod.
  - **<namespace>** is the namespace that contains the pod.

10.4. 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.

10.4.1. 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** object automatically.

重要

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

Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

Procedure

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: 1
     - name: test-network-1
       namespace: test-1
       type: Raw
       rawCNIConfig: '{
   ```

```
     "cniVersion": "0.3.1",
     "name": "test-network-1",
     "type": "bridge",
     "ipam": {
       "type": "static",
       "addresses": [
         {
           "address": "192.168.1.23/24"
         }
       ]
     }
    }'
```

1 Specify the configuration for the additional network attachment definition.

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

4. Confirm that the CNO created the NetworkAttachmentDefinition object by running the following command. Replace **<namespace>** with the namespace that you specified when configuring the network attachment. There might be a delay before the CNO creates the object.

```
$ oc get network-attachment-definitions -n <namespace>
```

Example output

```
NAME             AGE
test-network-1   14m
```

10.4.1.1. 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> 1
namespace: <namespace> 2
rawCNIConfig: '{ 3
  ...
}'
type: Raw
```

1. Specify a name for the additional network attachment that you are creating. The name must be unique within the specified **namespace**.

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

3. Specify the CNI plug-in configuration in JSON format, which is based on the following template.

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

bridge CNI plug-in JSON configuration object

```
{
  "cniVersion": "0.3.1",
  "name": "<name>", (1)
  "type": "bridge",
  "bridge": "<bridge>", (2)
  "ipam": { (3)
    ...
  },
  "ipMasq": false, (4)
  "isGateway": false, (5)
  "isDefaultGateway": false, (6)
  "forceAddress": false, (7)
  "hairpinMode": false, (8)
  "promiscMode": false, (9)
  "vlan": <vlan>, (10)
  "mtu": <mtu> (11)
}
```

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

2. Specify the name of the virtual bridge to use. If the bridge interface does not exist on the host, it is created. The default value is **cni0**.

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

4. Set to **true** to enable IP masquerading for traffic that leaves the virtual network. The source IP address for all traffic is rewritten to the bridge's IP address. If the bridge does not have an IP address, this setting has no effect. The default value is **false**.

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

6. Set to **true** to configure the bridge as the default gateway for the virtual network. The default value is **false**. If **isDefaultGateway** is set to **true**, then **isGateway** is also set to **true** automatically.

7. Set to **true** to allow assignment of a previously assigned IP address to the virtual bridge. When set to **false**, if an IPv4 address or an IPv6 address from overlapping subsets is assigned to the virtual bridge, an error occurs. The default value is **false**.

8. Set to **true** to allow the virtual bridge to send an ethernet frame back through the virtual port it was received on. This mode is also known as *reflective relay*. The default value is **false**.

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.

10.4.1.1.1. bridge configuration example

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

```
name: bridge-net
namespace: work-network
type: Raw
rawCNIConfig: '{ 1
  "cniVersion": "0.3.1",
  "name": "work-network",
  "type": "bridge",
  "isGateway": true,
  "vlan": 2,
  "ipam": {
    "type": "dhcp"
    }
}'
```

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

10.4.1.2. Configuration for ipam CNI plug-in

The ipam Container Network Interface (CNI) plug-in provides IP address management (IPAM) for other CNI plug-ins.

You can use the following methods for IP address assignment:

- Static assignment.
- Dynamic assignment through a DHCP server. The DHCP server you specify must be reachable from the additional network.
- Dynamic assignment through the Whereabouts IPAM CNI plug-in.

10.4.1.2.1. Static IP address assignment configuration

The following JSON describes the configuration for static IP address assignment:

Static assignment configuration

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

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

2. An IP address and network prefix that you specify. For example, if you specify **10.10.21.10/24**, then the additional network is assigned an IP address of **10.10.21.10** and the netmask is **255.255.255.0**.

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

4. An array describing routes to configure inside the pod.

5. The IP address range in CIDR format, such as **192.168.17.0/24**, or **0.0.0.0/0** for the default route.

6. The gateway where network traffic is routed.

7. Optional: DNS configuration.

8. An of array 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** is 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.

10.4.1.2.2. Dynamic IP address assignment configuration

The following JSON describes the configuration for dynamic IP address address assignment with DHCP.

RENEWAL OF DHCP LEASES

A pod obtains its original DHCP lease when it is created. The lease must be periodically renewed by a minimal DHCP server deployment running on the cluster.

To trigger the deployment of the DHCP server, you must create a shim network attachment by editing the Cluster Network Operator configuration, as in the following example:

Example shim network attachment definition

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  ...
  additionalNetworks:
  - name: dhcp-shim
    namespace: default
    type: Raw
    rawCNIConfig: |-
      {
        "name": "dhcp-shim",
        "cniVersion": "0.3.1",
        "type": "bridge",
        "ipam": {
          "type": "dhcp"
        }
      }
```

DHCP assignment configuration

```
{
  "ipam": {
    "type": "dhcp"
  }
}
```

10.4.1.2.3. Dynamic IP address assignment configuration with Whereabouts

The Whereabouts CNI plug-in allows the dynamic assignment of an IP address to an additional network without the use of a DHCP server.

The following JSON describes the configuration for dynamic IP address assignment with Whereabouts:

Whereabouts assignment configuration

```
{
  "ipam": {
    "type": "whereabouts",
    "range": "<range>", 1
  "exclude": ["<exclude_part>, ..."], 2
 }
}
```

1. Specify an IP address and range in CIDR notation. IP addresses are assigned from within this range of addresses.

2. Optional: Specify a list of IP addresses and ranges in CIDR notation. IP addresses within an excluded address range are not assigned.

10.4.1.2.4. Static IP address assignment configuration example

You can configure ipam for static IP address assignment:

```
{
 "ipam": {
  "type": "static",
   "addresses": [
    {
     "address": "191.168.1.7"
    }
   ]
 }
}
```

10.4.1.2.5. Dynamic IP address assignment configuration example using DHCP

You can configure ipam for DHCP:

```
{
 "ipam": {
  "type": "dhcp"
 }
}
```

10.4.1.2.6. Dynamic IP address assignment configuration example using Whereabouts

You can configure ipam to use Whereabouts:

```
{
 "ipam": {
  "type": "whereabouts",
  "range": "192.0.2.192/27",
  "exclude": [
    "192.0.2.192/30",
    "192.0.2.196/32"
  ]
 }
}
```

10.4.2. Next steps

- Attach a pod to an additional network .

10.5. 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 moves the specified network device from the network namespace of the host into the network namespace of the pod.

10.5.1. 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** object automatically.

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

Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

Procedure

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",
         "name": "test-network-1",
         "type": "host-device",
         "device": "eth1",
         "ipam": {
           "type": "static",
           "addresses": [
             {
   ```

```
            "address": "192.168.1.23/24"
          }
        ]
      }
    }'
```

1 Specify the configuration for the additional network attachment definition.

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

4. Confirm that the CNO created the NetworkAttachmentDefinition object by running the following command. Replace **<namespace>** with the namespace that you specified when configuring the network attachment. There might be a delay before the CNO creates the object.

   ```
   $ oc get network-attachment-definitions -n <namespace>
   ```

   Example output

   ```
   NAME                 AGE
   test-network-1       14m
   ```

10.5.1.1. 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> 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, the **default** namespace is used.

3 Specify the CNI plug-in configuration in JSON format, which is based on the following template.

重要

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>", 1
  "type": "host-device",
  "device": "<device>", 2
  "hwaddr": "<hwaddr>", 3
  "kernelpath": "<kernelpath>", 4
  "pciBusID": "<pciBusID>", 5
  "ipam": { 6
    ...
  }
}
```

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

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

3 Specify the device hardware MAC address.

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

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

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

10.5.1.1.1. host-device configuration example

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

```
name: hostdev-net
namespace: work-network
type: Raw
rawCNIConfig: '{ 1
  "cniVersion": "0.3.1",
  "name": "work-network",
  "type": "host-device",
  "device": "eth1",
  "ipam": {
    "type": "dhcp"
  }
}'
```

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

10.5.1.2. Configuration for ipam CNI plug-in

The ipam Container Network Interface (CNI) plug-in provides IP address management (IPAM) for other CNI plug-ins.

You can use the following methods for IP address assignment:

- Static assignment.
- Dynamic assignment through a DHCP server. The DHCP server you specify must be reachable from the additional network.
- Dynamic assignment through the Whereabouts IPAM CNI plug-in.

10.5.1.2.1. Static IP address assignment configuration

The following JSON describes the configuration for static IP address assignment:

Static assignment configuration

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

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

2 An IP address and network prefix that you specify. For example, if you specify **10.10.21.10/24**, then the additional network is assigned an IP address of **10.10.21.10** and the netmask is **255.255.255.0**.

3 The default gateway to route egress network traffic to.

4 An array describing routes to configure inside the pod.

5 The IP address range in CIDR format, such as **192.168.17.0/24**, or **0.0.0.0/0** for the default route.

6 The gateway where network traffic is routed.

7 Optional: DNS configuration.

8 An of array 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** is 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.

10.5.1.2.2. Dynamic IP address assignment configuration

The following JSON describes the configuration for dynamic IP address address assignment with DHCP.

RENEWAL OF DHCP LEASES

A pod obtains its original DHCP lease when it is created. The lease must be periodically renewed by a minimal DHCP server deployment running on the cluster.

To trigger the deployment of the DHCP server, you must create a shim network attachment by editing the Cluster Network Operator configuration, as in the following example:

Example shim network attachment definition

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  ...
  additionalNetworks:
  - name: dhcp-shim
    namespace: default
    type: Raw
    rawCNIConfig: |-
      {
        "name": "dhcp-shim",
        "cniVersion": "0.3.1",
        "type": "bridge",
        "ipam": {
          "type": "dhcp"
        }
      }
```

DHCP assignment configuration

```
{
  "ipam": {
    "type": "dhcp"
  }
}
```

10.5.1.2.3. Dynamic IP address assignment configuration with Whereabouts

The Whereabouts CNI plug-in allows the dynamic assignment of an IP address to an additional network without the use of a DHCP server.

The following JSON describes the configuration for dynamic IP address assignment with Whereabouts:

Whereabouts assignment configuration

```
{
 "ipam": {
  "type": "whereabouts",
  "range": "<range>", 1
  "exclude": ["<exclude_part>, ..."], 2
 }
}
```

1 Specify an IP address and range in CIDR notation. IP addresses are assigned from within this range of addresses.

2 Optional: Specify a list of IP addresses and ranges in CIDR notation. IP addresses within an excluded address range are not assigned.

10.5.1.2.4. Static IP address assignment configuration example

You can configure ipam for static IP address assignment:

```
{
 "ipam": {
  "type": "static",
   "addresses": [
    {
     "address": "191.168.1.7"
    }
   ]
 }
}
```

10.5.1.2.5. Dynamic IP address assignment configuration example using DHCP

You can configure ipam for DHCP:

```
{
 "ipam": {
  "type": "dhcp"
 }
}
```

10.5.1.2.6. Dynamic IP address assignment configuration example using Whereabouts

You can configure ipam to use Whereabouts:

```
{
  "ipam": {
    "type": "whereabouts",
    "range": "192.0.2.192/27",
    "exclude": [
       "192.0.2.192/30",
       "192.0.2.196/32"
    ]
  }
}
```

10.5.2. Next steps

- Attach a pod to an additional network .

10.6. 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.

10.6.1. 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** object automatically.

重要

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

Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

Procedure

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
    rawCNIConfig: '{
      "cniVersion": "0.3.1",
      "name": "test-network-1",
      "type": "ipvlan",
      "master": "eth1",
      "mode": "l2",
      "ipam": {
        "type": "static",
        "addresses": [
          {
            "address": "192.168.1.23/24"
          }
        ]
      }
    }'
```

1 Specify the configuration for the additional network attachment definition.

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

4. Confirm that the CNO created the NetworkAttachmentDefinition object by running the following command. Replace **<namespace>** with the namespace that you specified when configuring the network attachment. There might be a delay before the CNO creates the object.

```
$ oc get network-attachment-definitions -n <namespace>
```

Example output

```
NAME              AGE
test-network-1    14m
```

10.6.1.1. 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>", ①
  "type": "ipvlan",
  "mode": "<mode>", ②
  "master": "<master>", ③
  "mtu": <mtu>, ④
  "ipam": { ⑤
    ...
  }
}
```

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

② Specify the operating mode for the virtual network. The value must be **l2**, **l3**, or **l3s**. The default value is **l2**.

③ 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.

④ Set the maximum transmission unit (MTU) to the specified value. The default value is automatically set by the kernel.

⑤ Specify a configuration object for the ipam CNI plug-in. The plug-in manages IP address assignment for the attachment definition.

10.6.1.1.1. ipvlan configuration example

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

```
name: ipvlan-net
namespace: work-network
type: Raw
```

```
rawCNIConfig: '{ 1
  "cniVersion": "0.3.1",
  "name": "work-network",
  "type": "ipvlan",
  "master": "eth1",
  "mode": "l3",
  "ipam": {
    "type": "dhcp"
    }
}'
```

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

10.6.1.2. Configuration for ipam CNI plug-in

The ipam Container Network Interface (CNI) plug-in provides IP address management (IPAM) for other CNI plug-ins.

You can use the following methods for IP address assignment:

- Static assignment.
- Dynamic assignment through a DHCP server. The DHCP server you specify must be reachable from the additional network.
- Dynamic assignment through the Whereabouts IPAM CNI plug-in.

10.6.1.2.1. Static IP address assignment configuration

The following JSON describes the configuration for static IP address assignment:

Static assignment configuration

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

1. An array describing IP addresses to assign to the virtual interface. Both IPv4 and IPv6 IP addresses are supported.
2. An IP address and network prefix that you specify. For example, if you specify **10.10.21.10/24**, then the additional network is assigned an IP address of **10.10.21.10** and the netmask is **255.255.255.0**.
3. The default gateway to route egress network traffic to.
4. An array describing routes to configure inside the pod.
5. The IP address range in CIDR format, such as **192.168.17.0/24**, or **0.0.0.0/0** for the default route.
6. The gateway where network traffic is routed.
7. Optional: DNS configuration.
8. An of array 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** is 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.

10.6.1.2.2. Dynamic IP address assignment configuration

The following JSON describes the configuration for dynamic IP address address assignment with DHCP.

RENEWAL OF DHCP LEASES

A pod obtains its original DHCP lease when it is created. The lease must be periodically renewed by a minimal DHCP server deployment running on the cluster.

To trigger the deployment of the DHCP server, you must create a shim network attachment by editing the Cluster Network Operator configuration, as in the following example:

Example shim network attachment definition

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  ...
  additionalNetworks:
  - name: dhcp-shim
    namespace: default
    type: Raw
    rawCNIConfig: |-
      {
        "name": "dhcp-shim",
        "cniVersion": "0.3.1",
        "type": "bridge",
        "ipam": {
          "type": "dhcp"
        }
      }
```

DHCP assignment configuration

```
{
  "ipam": {
    "type": "dhcp"
  }
}
```

10.6.1.2.3. Dynamic IP address assignment configuration with Whereabouts

The Whereabouts CNI plug-in allows the dynamic assignment of an IP address to an additional network without the use of a DHCP server.

The following JSON describes the configuration for dynamic IP address assignment with Whereabouts:

Whereabouts assignment configuration

```
{
  "ipam": {
    "type": "whereabouts",
    "range": "<range>", 1
    "exclude": ["<exclude_part>, ..."], ②
  }
}
```

① Specify an IP address and range in CIDR notation. IP addresses are assigned from within this range of addresses.

② Optional: Specify a list of IP addresses and ranges in CIDR notation. IP addresses within an excluded address range are not assigned.

10.6.1.2.4. Static IP address assignment configuration example

You can configure ipam for static IP address assignment:

```
{
  "ipam": {
    "type": "static",
      "addresses": [
        {
          "address": "191.168.1.7"
        }
      ]
  }
}
```

10.6.1.2.5. Dynamic IP address assignment configuration example using DHCP

You can configure ipam for DHCP:

```
{
  "ipam": {
    "type": "dhcp"
  }
}
```

10.6.1.2.6. Dynamic IP address assignment configuration example using Whereabouts

You can configure ipam to use Whereabouts:

```
{
  "ipam": {
    "type": "whereabouts",
    "range": "192.0.2.192/27",
    "exclude": [
       "192.0.2.192/30",
       "192.0.2.196/32"
    ]
  }
}
```

10.6.2. Next steps

- Attach a pod to an additional network .

10.7. CONFIGURING A MACVLAN NETWORK WITH BASIC CUSTOMIZATIONS

As a cluster administrator, you can configure an additional network for your cluster using the macvlan Container Network Interface (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.

重要

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

You specify a basic configuration directly in YAML. This approach offers fewer configuration options than by specifying a macvlan configuration by using a CNI object directly in JSON.

10.7.1. 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** object automatically.

重要

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

Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

Procedure

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: 10.1.1.7/24
```

1 Specify the configuration for the additional network attachment definition.

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

4. Confirm that the CNO created the NetworkAttachmentDefinition object by running the following command. Replace **<namespace>** with the namespace that you specified when configuring the network attachment. There might be a delay before the CNO creates the object.

```
$ oc get network-attachment-definitions -n <namespace>
```

Example output

```
NAME                 AGE
test-network-1       14m
```

10.7.1.1. 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> 1
namespace: <namespace> 2
type: SimpleMacvlan
simpleMacvlanConfig:
  master: <master> 3
  mode: <mode> 4
  mtu: <mtu> 5
  ipamConfig: 6
    ...
```

1 Specify a name for the additional network attachment that you are creating. The name must be unique within the specified **namespace**.

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

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

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

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

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

10.7.1.1.1. 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
```

10.7.1.2. Configuration for ipam CNI plug-in

The ipam Container Network Interface (CNI) plug-in provides IP address management (IPAM) for other CNI plug-ins.

The following YAML configuration describes the parameters that you can set.

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.

10.7.1.2.1. 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 An IP address and network prefix that you specify. For example, if you specify **10.10.21.10/24**, then the additional network is assigned an IP address of **10.10.21.10** and the netmask is **255.255.255.0**.

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, such as **192.168.17.0/24**, or **0.0.0.0/0** for the default route.

6 The gateway where network traffic is routed.

7 Optional: The DNS configuration.

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** is 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.

10.7.1.2.2. Dynamic ipam configuration YAML

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

Dynamic ipam configuration YAML

```
ipamConfig:
  type: DHCP
```

10.7.1.2.3. 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
```

10.7.1.2.4. Dynamic IP address assignment configuration example

The following example shows an ipam configuration for DHCP:

```
ipamConfig:
  type: DHCP
```

10.7.2. Next steps

- Attach a pod to an additional network .

10.8. CONFIGURING A MACVLAN NETWORK

As a cluster administrator, you can configure an additional network for your cluster using the macvlan Container Network Interface (CNI) plug-in with advanced customization. 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.

重要

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

You specify a configuration with a CNI object. This approach allows you to specify additional configuration options that are not available when using a YAML configuration.

10.8.1. 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** object automatically.

重要

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

Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

Procedure

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: Raw
    rawCNIConfig: '{
      "cniVersion": "0.3.1",
      "name": "test-network-1",
      "type": "macvlan",
      "master": "eth1",
      "ipam": {
        "type": "static",
        "addresses": [
          {
            "address": "192.168.1.23/24"
          }
        ]
      }
    }'
```

1 Specify the configuration for the additional network attachment definition.

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

4. Confirm that the CNO created the NetworkAttachmentDefinition object by running the following command. Replace **<namespace>** with the namespace that you specified when configuring the network attachment. There might be a delay before the CNO creates the object.

```
$ oc get network-attachment-definitions -n <namespace>
```

Example output

```
NAME             AGE
test-network-1   14m
```

10.8.1.1. Configuration for macvlan CNI plug-in

The configuration for an additional network attachment that uses the macvlan 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 macvlan CNI plug-in:

macvlan CNI plug-in JSON configuration object

```
{
  "cniVersion": "0.3.1",
  "name": "<name>", ①
  "type": "macvlan",
  "mode": "<mode>", ②
  "master": "<master>", ③
  "mtu": <mtu>, ④
  "ipam": { ⑤
    ...
  }
}
```

① Specify a name for the additional network attachment that you are creating. The name must be unique within the specified **namespace**.

② Configures traffic visibility on the virtual network. Must be either **bridge**, **passthru**, **private**, or **vepa**. If a value is not provided, the default value is **bridge**.

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

④ Set the maximum transmission unit (MTU) to the specified value. The default value is automatically set by the kernel.

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

10.8.1.1.1. macvlan configuration example

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

```
name: macvlan-net
namespace: work-network
type: Raw
rawCNIConfig: |-
  {
    "cniVersion": "0.3.1",
    "name": "macvlan-net",
    "type": "macvlan",
    "master": "eth1",
    "mode": "bridge",
    "ipam": {
      "type": "dhcp"
      }
  }
```

10.8.1.2. Configuration for ipam CNI plug-in

The ipam Container Network Interface (CNI) plug-in provides IP address management (IPAM) for other CNI plug-ins.

You can use the following methods for IP address assignment:

- Static assignment.
- Dynamic assignment through a DHCP server. The DHCP server you specify must be reachable from the additional network.
- Dynamic assignment through the Whereabouts IPAM CNI plug-in.

10.8.1.2.1. Static IP address assignment configuration

The following JSON describes the configuration for static IP address assignment:

Static assignment configuration

```
{
  "ipam": {
    "type": "static",
    "addresses": [ 1
      {
        "address": "<address>", 2
        "gateway": "<gateway>" 3
      }
    ],
    "routes": [ 4
      {
```

```
    "dst": "<dst>", 5
    "gw": "<gw>" 6
   }
  ],
  "dns": { 7
   "nameservers": ["<nameserver>"], 8
   "domain": "<domain>", 9
   "search": ["<search_domain>"] 10
  }
 }
}
```

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

2 An IP address and network prefix that you specify. For example, if you specify **10.10.21.10/24**, then the additional network is assigned an IP address of **10.10.21.10** and the netmask is **255.255.255.0**.

3 The default gateway to route egress network traffic to.

4 An array describing routes to configure inside the pod.

5 The IP address range in CIDR format, such as **192.168.17.0/24**, or **0.0.0.0/0** for the default route.

6 The gateway where network traffic is routed.

7 Optional: DNS configuration.

8 An of array 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** is 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.

10.8.1.2.2. Dynamic IP address assignment configuration

The following JSON describes the configuration for dynamic IP address address assignment with DHCP.

RENEWAL OF DHCP LEASES

A pod obtains its original DHCP lease when it is created. The lease must be periodically renewed by a minimal DHCP server deployment running on the cluster.

To trigger the deployment of the DHCP server, you must create a shim network attachment by editing the Cluster Network Operator configuration, as in the following example:

Example shim network attachment definition

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  ...
  additionalNetworks:
  - name: dhcp-shim
    namespace: default
    type: Raw
    rawCNIConfig: |-
      {
        "name": "dhcp-shim",
        "cniVersion": "0.3.1",
        "type": "bridge",
        "ipam": {
          "type": "dhcp"
        }
      }
```

DHCP assignment configuration

```
{
  "ipam": {
    "type": "dhcp"
  }
}
```

10.8.1.2.3. Dynamic IP address assignment configuration with Whereabouts

The Whereabouts CNI plug-in allows the dynamic assignment of an IP address to an additional network without the use of a DHCP server.

The following JSON describes the configuration for dynamic IP address assignment with Whereabouts:

Whereabouts assignment configuration

```
{
  "ipam": {
    "type": "whereabouts",
    "range": "<range>", 1
    "exclude": ["<exclude_part>, ..."], 2
  }
}
```

1 Specify an IP address and range in CIDR notation. IP addresses are assigned from within this range of addresses.

2 Optional: Specify a list of IP addresses and ranges in CIDR notation. IP addresses within an excluded address range are not assigned.

10.8.1.2.4. Static IP address assignment configuration example

You can configure ipam for static IP address assignment:

```
{
  "ipam": {
    "type": "static",
      "addresses": [
        {
          "address": "191.168.1.7"
        }
      ]
  }
}
```

10.8.1.2.5. Dynamic IP address assignment configuration example using DHCP

You can configure ipam for DHCP:

```
{
  "ipam": {
    "type": "dhcp"
  }
}
```

10.8.1.2.6. Dynamic IP address assignment configuration example using Whereabouts

You can configure ipam to use Whereabouts:

```
{
  "ipam": {
    "type": "whereabouts",
    "range": "192.0.2.192/27",
    "exclude": [
       "192.0.2.192/30",
       "192.0.2.196/32"
    ]
  }
}
```

10.8.2. Next steps

- Attach a pod to an additional network .

10.9. EDITING AN ADDITIONAL NETWORK

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

10.9.1. Modifying an additional network attachment definition

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

Prerequisites

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

Procedure

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

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

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

2. In the **additionalNetworks** collection, update the additional network with your changes.

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

4. Optional: Confirm that the CNO updated the **NetworkAttachmentDefinition** object by running the following command. Replace **<network-name>** with the name of the additional network to display. There might be a delay before the CNO updates the **NetworkAttachmentDefinition** object to reflect your changes.

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

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

```
$ oc get network-attachment-definitions net1 -o go-template='{{printf "%s\n" .spec.config}}'
{ "cniVersion": "0.3.1", "type": "macvlan",
"master": "ens5",
"mode": "bridge",
"ipam":       {"type":"static","routes":[{"dst":"0.0.0.0/0","gw":"10.128.2.1"}],"addresses":
[{"address":"10.128.2.100/23","gateway":"10.128.2.1"}],"dns":{"nameservers":
["172.30.0.10"],"domain":"us-west-2.compute.internal","search":["us-west-
2.compute.internal"]}} }
```

10.10. REMOVING AN ADDITIONAL NETWORK

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

10.10.1. Removing an additional network attachment definition

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

Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

Procedure

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

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

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

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

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

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

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

4. Optional: Confirm that the additional network CR was deleted by running the following command:

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

第 11 章 HARDWARE NETWORKS

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

The Single Root I/O Virtualization (SR-IOV) specification is a standard for a type of PCI device assignment that can share a single device with multiple pods.

SR-IOV enables you to segment a compliant network device, recognized on the host node as a physical function (PF), into multiple virtual functions (VFs). The VF is used like any other network device. The SR-IOV device driver for the device determines how the VF is exposed in the container:

- **netdevice** driver: A regular kernel network device in the **netns** of the container
- **vfio-pci** driver: A character device mounted in the container

You can use SR-IOV network devices with additional networks on your OpenShift Container Platform cluster for application that require high bandwidth or low latency.

11.1.1. Components that manage SR-IOV network devices

The SR-IOV Network Operator creates and manages the components of the SR-IOV stack. It performs the following functions:

- Orchestrates discovery and management of SR-IOV network devices
- Generates **NetworkAttachmentDefinition** custom resources for the SR-IOV Container Network Interface (CNI)
- Creates and updates the configuration of the SR-IOV network device plug-in
- Creates node specific **SriovNetworkNodeState** custom resources
- Updates the **spec.interfaces** field in each **SriovNetworkNodeState** custom resource

The Operator provisions the following components:

SR-IOV network configuration daemon

A DaemonSet that is deployed on worker nodes when the SR-IOV Operator starts. The daemon is responsible for discovering and initializing SR-IOV network devices in the cluster.

SR-IOV Operator webhook

A dynamic admission controller webhook that validates the Operator custom resource and sets appropriate default values for unset fields.

SR-IOV Network resources injector

A dynamic admission controller webhook that provides functionality for patching Kubernetes pod specifications with requests and limits for custom network resources such as SR-IOV VFs.

SR-IOV network device plug-in

A device plug-in that discovers, advertises, and allocates 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 that the scheduler can schedule pods on nodes with sufficient resources.

SR-IOV CNI plug-in

A CNI plug-in that attaches VF interfaces allocated from the SR-IOV device plug-in directly into a pod.

注意

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

11.1.1.1. Supported devices

OpenShift Container Platform supports the following Network Interface Card (NIC) models:

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

11.1.1.2. Automated discovery of SR-IOV network devices

The SR-IOV Network Operator searches your cluster for SR-IOV capable network devices on worker nodes. The Operator creates and updates a SriovNetworkNodeState custom resource (CR) for each worker node that provides a compatible SR-IOV network device.

The CR is assigned the same name as the worker node. The **status.interfaces** list provides information about the network devices on a node.

重要

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

11.1.1.2.1. Example SriovNetworkNodeState object

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

An SriovNetworkNodeState object

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodeState
metadata:
  name: node-25 1
  namespace: openshift-sriov-network-operator
  ownerReferences:
  - apiVersion: sriovnetwork.openshift.io/v1
    blockOwnerDeletion: true
    controller: true
    kind: SriovNetworkNodePolicy
    name: default
spec:
  dpConfigVersion: "39824"
status:
  interfaces: 2
  - deviceID: "1017"
    driver: mlx5_core
    mtu: 1500
    name: ens785f0
    pciAddress: "0000:18:00.0"
    totalvfs: 8
    vendor: 15b3
  - deviceID: "1017"
    driver: mlx5_core
    mtu: 1500
    name: ens785f1
    pciAddress: "0000:18:00.1"
    totalvfs: 8
    vendor: 15b3
  - deviceID: 158b
    driver: i40e
    mtu: 1500
    name: ens817f0
    pciAddress: 0000:81:00.0
    totalvfs: 64
    vendor: "8086"
  - deviceID: 158b
    driver: i40e
    mtu: 1500
    name: ens817f1
    pciAddress: 0000:81:00.1
    totalvfs: 64
    vendor: "8086"
  - deviceID: 158b
    driver: i40e
    mtu: 1500
    name: ens803f0
    pciAddress: 0000:86:00.0
    totalvfs: 64
    vendor: "8086"
  syncStatus: Succeeded
```

1 The value of the **name** field is the same as the name of the worker node.

2 The **interfaces** stanza includes a list of all of the SR-IOV devices discovered by the Operator on the worker node.

11.1.1.3. Example use of a virtual function in a pod

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

This example shows a pod using a virtual function (VF) in RDMA mode:

Pod spec that uses RDMA mode

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

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

Pod spec that uses DPDK mode

```
apiVersion: v1
kind: Pod
metadata:
  name: dpdk-app
  annotations:
    k8s.v1.cni.cncf.io/networks: sriov-dpdk-net
spec:
  containers:
  - name: testpmd
    image: <DPDK_image>
    securityContext:
     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.

11.1.2. Next steps

- Installing the SR-IOV Network Operator
- Optional: Configuring the SR-IOV Network Operator
- Configuring an SR-IOV network device
- If you use OpenShift Virtualization: Configuring an SR-IOV network device for virtual machines
- Configuring an SR-IOV network attachment
- Adding a pod to an SR-IOV additional network

11.2. 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.

11.2.1. Installing SR-IOV Network Operator

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

11.2.1.1. CLI: Installing the SR-IOV Network Operator

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

Prerequisites

- A cluster installed on bare-metal hardware with nodes that have hardware that supports SR-IOV.
- Install the OpenShift CLI (**oc**).
- An account with **cluster-admin** privileges.

Procedure

1. To create the **openshift-sriov-network-operator** namespace, enter the following command:

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

```
metadata:
  name: openshift-sriov-network-operator
EOF
```

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

```
$ cat << EOF| oc create -f -
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: sriov-network-operators
  namespace: openshift-sriov-network-operator
spec:
  targetNamespaces:
  - openshift-sriov-network-operator
EOF
```

3. Subscribe to the SR-IOV Network Operator.

    a. Run the following command to get the OpenShift Container Platform major and minor version. It is required for the **channel** value in the next step.

    ```
    $ OC_VERSION=$(oc version -o yaml | grep openshiftVersion | \
        grep -o '[0-9]*[.][0-9]*' | head -1)
    ```

    b. To create a Subscription CR for the SR-IOV Network Operator, enter the following command:

    ```
    $ cat << EOF| oc create -f -
    apiVersion: operators.coreos.com/v1alpha1
    kind: Subscription
    metadata:
      name: sriov-network-operator-subsription
      namespace: openshift-sriov-network-operator
    spec:
      channel: "${OC_VERSION}"
      name: sriov-network-operator
      source: redhat-operators
      sourceNamespace: openshift-marketplace
    EOF
    ```

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

```
$ oc get csv -n openshift-sriov-network-operator \
  -o custom-columns=Name:.metadata.name,Phase:.status.phase
```

Example output

```
Name                                       Phase
sriov-network-operator.4.4.0-202006160135   Succeeded
```

11.2.1.2. Web console: Installing the SR-IOV Network Operator

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

> **注意**
>
> You must create the operator group by using the CLI.

Prerequisites

- A cluster installed on bare-metal hardware with nodes that have hardware that supports SR-IOV.
- Install the OpenShift CLI (**oc**).
- An account with **cluster-admin** privileges.

Procedure

1. Create a namespace for the SR-IOV Network Operator:
   a. In the OpenShift Container Platform web console, click **Administration → Namespaces**.
   b. Click **Create Namespace**.
   c. In the **Name** field, enter **openshift-sriov-network-operator**, and then click **Create**.
2. Install the SR-IOV Network Operator:
   a. In the OpenShift Container Platform web console, click **Operators → OperatorHub**.
   b. Select **SR-IOV Network Operator** from the list of available Operators, and then click **Install**.
   c. On the **Install Operator** page, under **A specific namespace on the cluster**, select **openshift-sriov-network-operator**.
   d. Click **Install**.
3. Verify that the SR-IOV Network Operator is installed successfully:
   a. Navigate to the **Operators → Installed Operators** page.
   b. Ensure that **SR-IOV Network Operator** is listed in the **openshift-sriov-network-operator** project with a **Status** of **InstallSucceeded**.

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

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

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

- Navigate to the **Workloads** → **Pods** page and check the logs for pods in the **openshift-sriov-network-operator** project.

11.2.2. Next steps

- Optional: Configuring the SR-IOV Network Operator

11.3. 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.

11.3.1. Configuring the SR-IOV Network Operator

重要

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.

注意

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** object provides several fields for configuring the operator:

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

11.3.1.1. 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 daemon set 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
```

Example output

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

11.3.1.2. 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 daemon set 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
```

Example output

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

11.3.1.3. 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.

11.3.1.4. 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 CLI (**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> } }'
```

11.3.1.5. 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 CLI (**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> } }'
```

11.3.1.6. 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.

重要

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": ""**.

11.3.2. Next steps

- Configuring an SR-IOV network device

11.4. CONFIGURING AN SR-IOV NETWORK DEVICE

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

11.4.1. SR-IOV network node configuration object

You specify the SR-IOV network device configuration for a node by defining an **SriovNetworkNodePolicy** object. The object is part of the **sriovnetwork.openshift.io** API group.

The following YAML describes an **SriovNetworkNodePolicy** object:

```
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 The name for the CR object.

② The namespace where the SR-IOV Operator is installed.

③ The resource name of the SR-IOV device plug-in. You can create multiple **SriovNetworkNodePolicy** objects for a resource name.

4. The node selector to select which nodes are configured. Only SR-IOV network devices on selected nodes are configured. The SR-IOV Container Network Interface (CNI) plug-in and device plug-in are deployed on only selected nodes.

5. Optional: An integer value between **0** and **99**. A smaller number gets higher priority, so a priority of **10** is higher than a priority of **99**. The default value is **99**.

6. Optional: The maximum transmission unit (MTU) of the virtual function. The maximum MTU value can vary for different NIC models.

7. 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 device for the Operator to configure. You do not have to specify values for all the parameters. It is recommended to identify the network device with enough precision to avoid selecting a device unintentionally. If you specify **rootDevices**, you must also specify a value for **vendor**, **deviceID**, or **pfNames**. If you specify both **pfNames** and **rootDevices** at the same time, ensure that they point to the same device.

9. Optional: The vendor hex code of the SR-IOV network device. The only allowed values are **8086** and **15b3**.

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

11. Optional: An array of one or more physical function (PF) names for the device.

12. An array of one or more PCI bus addresses for the PF of the device. Provide the address in the following format: **0000:02:00.1**.

13. Optional: The driver type for the virtual functions. The only allowed values are **netdevice** and **vfio-pci**. The default value is **netdevice**.

    注意

    For a Mellanox card to work in Data Plane Development Kit (DPDK) mode on bare metal nodes, use the **netdevice** driver type and set **isRdma** to **true**.

14. Optional: Whether to enable remote direct memory access (RDMA) mode. The default value is **false**.

    注意

    If the **isRDMA** parameter is set to **true**, you can continue to use the RDMA enabled VF as a normal network device. A device can be used in either mode.

11.4.1.1. Virtual function (VF) partitioning for SR-IOV devices

In some cases, you might want to split virtual functions (VFs) from the same physical function (PF) into multiple resource pools. For example, you might want some of the VFs to load with the default driver and the remaining VFs load with the **vfio-pci** driver. In such a deployment, the **pfNames** selector in your SriovNetworkNodePolicy custom resource (CR) can be used to specify a range of VFs for a pool using the following format: **<pfname>#<first_vf>-<last_vf>**.

For example, the following YAML shows the selector for an interface named **netpf0** with VF **2** through **7**:

```
pfNames: ["netpf0#2-7"]
```

- **netpf0** is the PF interface name.
- **2** is the first VF index (0-based) that is included in the range.
- **7** is the last VF index (0-based) that is included in the range.

You can select VFs from the same PF by using different policy CRs if the following requirements are met:

- The **numVfs** value must be identical for policies that select the same PF.
- The VF index must be in the range of **0** to **<numVfs>-1**. For example, if you have a policy with **numVfs** set to **8**, then the **<first_vf>** value must not be smaller than **0**, and the **<last_vf>** must not be larger than **7**.
- The VFs ranges in different policies must not overlap.
- The **<first_vf>** must not be larger than the **<last_vf>**.

The following example illustrates NIC partitioning for an SR-IOV device.

The policy **policy-net-1** defines a resource pool **net-1** that contains the VF **0** of PF **netpf0** with the default VF driver. The policy **policy-net-1-dpdk** defines a resource pool **net-1-dpdk** that contains the VF **8** to **15** of PF **netpf0** with the **vfio** VF driver.

Policy **policy-net-1**:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: policy-net-1
  namespace: openshift-sriov-network-operator
spec:
  resourceName: net1
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  numVfs: 16
  nicSelector:
    pfNames: ["netpf0#0-0"]
  deviceType: netdevice
```

Policy **policy-net-1-dpdk**:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: policy-net-1-dpdk
  namespace: openshift-sriov-network-operator
spec:
  resourceName: net1dpdk
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
```

```
numVfs: 16
nicSelector:
  pfNames: ["netpf0#8-15"]
deviceType: vfio-pci
```

11.4.2. Configuring SR-IOV network devices

The SR-IOV Network Operator adds the **SriovNetworkNodePolicy.sriovnetwork.openshift.io** CustomResourceDefinition to OpenShift Container Platform. You can configure an SR-IOV network device by creating a SriovNetworkNodePolicy custom resource (CR).

注意

When applying the configuration specified in a **SriovNetworkNodePolicy** object, the SR-IOV Operator might drain the nodes, and in some cases, reboot nodes.

It might take several minutes for a configuration change to apply.

Prerequisites

- You installed the OpenShift CLI (**oc**).
- You have access to the cluster as a user with the **cluster-admin** role.
- You have installed the SR-IOV Network Operator.
- You have enough available nodes in your cluster to handle the evicted workload from drained nodes.
- You have not selected any control plane nodes for SR-IOV network device configuration.

Procedure

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

2. Create the **SriovNetworkNodePolicy** object:

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

   where **<name>** specifies the name for this configuration.

   After applying the configuration update, all the pods in **sriov-network-operator** namespace transition to the **Running** status.

3. To verify that the SR-IOV network device is configured, enter the following command. Replace **<node_name>** with the name of a node with the SR-IOV network device that you just configured.

   ```
   $ oc get sriovnetworknodestates -n openshift-sriov-network-operator <node_name> -o
   jsonpath='{.status.syncStatus}'
   ```

11.4.3. Next steps

- Configuring an SR-IOV network attachment

11.5. CONFIGURING AN SR-IOV ETHERNET NETWORK ATTACHMENT

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

11.5.1. Ethernet device configuration object

You can configure an Ethernet network device by defining an **SriovNetwork** object.

The following YAML describes an **SriovNetwork** object:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: <name> ①
  namespace: openshift-sriov-network-operator ②
spec:
  resourceName: <sriov_resource_name> ③
  networkNamespace: <target_namespace> ④
  vlan: <vlan> ⑤
  spoofChk: "<spoof_check>" ⑥
  ipam: |- ⑦
    {}
  linkState: <link_state> ⑧
  maxTxRate: <max_tx_rate> ⑨
  minTxRate: <min_tx_rate> ⑩
  vlanQoS: <vlan_qos> ⑪
  trust: "<trust_vf>" ⑫
  capabilities: <capabilities> ⑬
```

① A name for the object. The SR-IOV Network Operator creates a **NetworkAttachmentDefinition** object with same name.

② The namespace where the SR-IOV Network Operator is installed.

③ The value for the **spec.resourceName** parameter from the **SriovNetworkNodePolicy** object that defines the SR-IOV hardware for this additional network.

④ The target namespace for the **SriovNetwork** object. Only pods in the target namespace can attach to the additional network.

⑤ Optional: A Virtual LAN (VLAN) ID for the additional network. The integer value must be from **0** to **4095**. The default value is **0**.

⑥ Optional: The spoof check mode of the VF. The allowed values are the strings **"on"** and **"off"**.

重要

You must enclose the value you specify in quotes or the object is rejected by the SR-IOV Network Operator.

7 A configuration object for the IPAM CNI plug-in as a YAML block scalar. The plug-in manages IP address assignment for the attachment definition.

8 Optional: The link state of virtual function (VF). Allowed value are **enable**, **disable** and **auto**.

9 Optional: A maximum transmission rate, in Mbps, for the VF.

10 Optional: A minimum transmission rate, in Mbps, for the VF. This value must be less than or equal to the maximum transmission rate.

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

11 Optional: An IEEE 802.1p priority level for the VF. The default value is **0**.

12 Optional: The trust mode of the VF. The allowed values are the strings **"on"** and **"off"**.

> **重要**
>
> You must enclose the value that you specify in quotes, or the SR-IOV Network Operator rejects the object.

13 Optional: The capabilities to configure for this additional network. You can specify **"{ "ips": true }"** to enable IP address support or **"{ "mac": true }"** to enable MAC address support.

11.5.1.1. Configuration for ipam CNI plug-in

The ipam Container Network Interface (CNI) plug-in provides IP address management (IPAM) for other CNI plug-ins.

You can use the following methods for IP address assignment:

- Static assignment.
- Dynamic assignment through a DHCP server. The DHCP server you specify must be reachable from the additional network.
- Dynamic assignment through the Whereabouts IPAM CNI plug-in.

11.5.1.1.1. Static IP address assignment configuration

The following JSON describes the configuration for static IP address assignment:

Static assignment configuration

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

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

2 An IP address and network prefix that you specify. For example, if you specify **10.10.21.10/24**, then the additional network is assigned an IP address of **10.10.21.10** and the netmask is **255.255.255.0**.

3 The default gateway to route egress network traffic to.

4 An array describing routes to configure inside the pod.

5 The IP address range in CIDR format, such as **192.168.17.0/24**, or **0.0.0.0/0** for the default route.

6 The gateway where network traffic is routed.

7 Optional: DNS configuration.

8 An of array 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** is 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.

11.5.1.1.2. Dynamic IP address assignment configuration

The following JSON describes the configuration for dynamic IP address address assignment with DHCP.

RENEWAL OF DHCP LEASES

A pod obtains its original DHCP lease when it is created. The lease must be periodically renewed by a minimal DHCP server deployment running on the cluster.

The SR-IOV Network Operator does not create a DHCP server deployment; The Cluster Network Operator is responsible for creating the minimal DHCP server deployment.

To trigger the deployment of the DHCP server, you must create a shim network attachment by editing the Cluster Network Operator configuration, as in the following example:

Example shim network attachment definition

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  ...
  additionalNetworks:
  - name: dhcp-shim
    namespace: default
    type: Raw
    rawCNIConfig: |-
      {
        "name": "dhcp-shim",
        "cniVersion": "0.3.1",
        "type": "bridge",
        "ipam": {
          "type": "dhcp"
        }
      }
```

DHCP assignment configuration

```
{
  "ipam": {
    "type": "dhcp"
  }
}
```

11.5.1.1.3. Dynamic IP address assignment configuration with Whereabouts

The Whereabouts CNI plug-in allows the dynamic assignment of an IP address to an additional network without the use of a DHCP server.

The following JSON describes the configuration for dynamic IP address assignment with Whereabouts:

Whereabouts assignment configuration

```
{
  "ipam": {
    "type": "whereabouts",
  "range": "<range>", 1
  "exclude": ["<exclude_part>, ..."], 2
 }
}
```

1 Specify an IP address and range in CIDR notation. IP addresses are assigned from within this range of addresses.

2 Optional: Specify a list of IP addresses and ranges in CIDR notation. IP addresses within an excluded address range are not assigned.

11.5.1.1.4. Static IP address assignment configuration example

You can configure ipam for static IP address assignment:

```
{
 "ipam": {
  "type": "static",
   "addresses": [
    {
     "address": "191.168.1.7"
    }
   ]
 }
}
```

11.5.1.1.5. Dynamic IP address assignment configuration example using DHCP

You can configure ipam for DHCP:

```
{
 "ipam": {
  "type": "dhcp"
 }
}
```

11.5.1.1.6. Dynamic IP address assignment configuration example using Whereabouts

You can configure ipam to use Whereabouts:

```
{
 "ipam": {
  "type": "whereabouts",
  "range": "192.0.2.192/27",
  "exclude": [
   "192.0.2.192/30",
   "192.0.2.196/32"
  ]
 }
}
```

11.5.2. Configuring SR-IOV additional network

You can configure an additional network that uses SR-IOV hardware by creating a **SriovNetwork** object. When you create a **SriovNetwork** object, the SR-IOV Operator automatically creates a **NetworkAttachmentDefinition** object.

注意

Do not modify or delete a **SriovNetwork** object if it is attached to any pods in the **running** state.

Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

Procedure

1. Create a **SriovNetwork** object, and then save the YAML in the **<name>.yaml** file, where **<name>** is a name for this additional network. The object specification might resemble the following example:

   ```
   apiVersion: sriovnetwork.openshift.io/v1
   kind: SriovNetwork
   metadata:
     name: attach1
     namespace: openshift-sriov-network-operator
   spec:
     resourceName: net1
     networkNamespace: project2
     ipam: |-
       {
         "type": "host-local",
         "subnet": "10.56.217.0/24",
         "rangeStart": "10.56.217.171",
         "rangeEnd": "10.56.217.181",
         "gateway": "10.56.217.1"
       }
   ```

2. To create the object, enter the following command:

   ```
   $ oc create -f <name>.yaml
   ```

   where **<name>** specifies the name of the additional network.

3. Optional: To confirm that the **NetworkAttachmentDefinition** object that is associated with the **SriovNetwork** object that you created in the previous step exists, enter the following command. Replace **<namespace>** with the networkNamespace you specified in the **SriovNetwork** object.

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

11.5.3. Next steps

- Adding a pod to an SR-IOV additional network

11.5.4. Additional resources

- Configuring an SR-IOV network device

11.6. 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.

11.6.1. Runtime configuration for a network attachment

When attaching a pod to an additional network, you can specify a runtime configuration to make specific customizations for the pod. For example, you can request a specific MAC hardware address.

You specify the runtime configuration by setting an annotation in the pod specification. The annotation key is **k8s.v1.cni.cncf.io/networks**, and it accepts a JSON object that describes the runtime configuration.

11.6.1.1. Runtime configuration for an Ethernet-based SR-IOV attachment

The following JSON describes the runtime configuration options for an Ethernet-based SR-IOV network attachment.

```
[
  {
    "name": "<name>", 1
    "mac": "<mac_address>", 2
    "ips": ["<cidr_range>"] 3
  }
]
```

1 The name of the SR-IOV network attachment definition CR.

2 Optional: The MAC address for the SR-IOV device that is allocated from the resource type defined in the SR-IOV network attachment definition CR. To use this feature, you also must specify **{ "mac": true }** in the **SriovNetwork** object.

3 Optional: IP addresses for the SR-IOV device that is allocated from the resource type defined in the SR-IOV network attachment definition CR. Both IPv4 and IPv6 addresses are supported. To use this feature, you also must specify **{ "ips": true }** in the **SriovNetwork** object.

Example runtime configuration

```
apiVersion: v1
kind: Pod
metadata:
  name: sample-pod
  annotations:
    k8s.v1.cni.cncf.io/networks: |-
      [
        {
          "name": "net1",
          "mac": "20:04:0f:f1:88:01",
          "ips": ["192.168.10.1/24", "2001::1/64"]
        }
      ]
spec:
  containers:
  - name: sample-container
    image: <image>
    imagePullPolicy: IfNotPresent
    command: ["sleep", "infinity"]
```

11.6.2. Adding a pod to an additional network

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

When a pod is created additional networks are attached to it. However, if a pod already exists, you cannot attach additional networks to it.

The pod must be in the same namespace as the additional network.

注意

If a network attachment is managed by the SR-IOV Network Operator, the SR-IOV Network Resource Injector adds the **resource** field to the **Pod** object automatically.

重要

When specifying an SR-IOV hardware network for a **Deployment** object or a **ReplicationController** object, you must specify the namespace of the **NetworkAttachmentDefinition** object. For more information, see the following bugs: BZ#1846333 and BZ#1840962.

Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in to the cluster.
- Install the SR-IOV Operator.
- Create an **SriovNetwork** object to attach the pod to.

Procedure

1. Add an annotation to the **Pod** object. Only one of the following annotation formats can be used:
   a. To attach an additional network without any customization, add an annotation with the following format. Replace **<network>** with the name of the additional network to associate with the pod:

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

1 To specify more than one additional network, separate each network with a comma. Do not include whitespace between the comma. If you specify the same additional

b. To attach an additional network with customizations, add an annotation with the following format:

```
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: |-
      [
        {
          "name": "<network>", 1
          "namespace": "<namespace>", 2
          "default-route": ["<default-route>"] 3
        }
      ]
```

1 Specify the name of the additional network defined by a **NetworkAttachmentDefinition** object.

2 Specify the namespace where the **NetworkAttachmentDefinition** object is defined.

3 Optional: Specify an override for the default route, such as **192.168.17.1**.

2. To create the pod, enter the following command. Replace **<name>** with the name of the pod.

```
$ oc create -f <name>.yaml
```

3. Optional: To Confirm that the annotation exists in the **Pod** CR, enter the following command, replacing **<name>** with the name of the pod.

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

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

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

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

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

Prerequisites

- Install the OpenShift CLI (**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 an SR-IOV pod spec:

```
apiVersion: v1
kind: Pod
metadata:
 name: sample-pod
 annotations:
  k8s.v1.cni.cncf.io/networks: <name> [1]
spec:
 containers:
 - name: sample-container
  image: <image> [2]
  command: ["sleep", "infinity"]
  resources:
   limits:
    memory: "1Gi" [3]
    cpu: "2" [4]
   requests:
    memory: "1Gi"
    cpu: "2"
```

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

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

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

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

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

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

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

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

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

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

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

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

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

11.6.4. Additional resources

- Configuring an SR-IOV Ethernet network attachment

11.7. USING HIGH PERFORMANCE MULTICAST

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

11.7.1. Configuring high performance multicast

The OpenShift SDN default Container Network Interface (CNI) network provider supports multicast between pods on the default network. This is best used for low-bandwidth coordination or service discovery, and not high-bandwidth applications. For applications such as streaming media, like Internet Protocol television (IPTV) and multipoint videoconferencing, you can utilize Single Root I/O Virtualization (SR-IOV) hardware to provide near-native performance.

When using additional SR-IOV interfaces for multicast:

- Multicast packages must be sent or received by a pod through the additional SR-IOV interface.

- The physical network which connects the SR-IOV interfaces decides the multicast routing and topology, which is not controlled by OpenShift Container Platform.

11.7.2. Using an SR-IOV interface for multicast

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

Prerequisites

- Install the OpenShift CLI (**oc**).
- You must log in to the cluster with a user that has the **cluster-admin** role.

Procedure

1. Create a **SriovNetworkNodePolicy** object:

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

2. Create a **SriovNetwork** object:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: net-example
  namespace: openshift-sriov-network-operator
spec:
  networkNamespace: default
  ipam: | 1
    {
      "type": "host-local", 2
      "subnet": "10.56.217.0/24",
      "rangeStart": "10.56.217.171",
      "rangeEnd": "10.56.217.181",
      "routes": [
        {"dst": "224.0.0.0/5"},
        {"dst": "232.0.0.0/5"}
      ],
      "gateway": "10.56.217.1"
    }
  resourceName: example
```

1 2 If you choose to configure DHCP as IPAM, ensure that you provision the following default routes through your DHCP server: **224.0.0.0/5** and **232.0.0.0/5**. This is to override the static multicast route set by the default network provider.

3. Create a pod with multicast application:

```
apiVersion: v1
kind: Pod
metadata:
  name: testpmd
  namespace: default
  annotations:
    k8s.v1.cni.cncf.io/networks: nic1
spec:
  containers:
  - name: example
    image: rhel7:latest
    securityContext:
      capabilities:
        add: ["NET_ADMIN"] 1
    command: [ "sleep", "infinity"]
```

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

11.8. 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).

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

重要

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/.

重要

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/.

11.8.2. Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.
- You must have installed the SR-IOV Network Operator.

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

Procedure

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

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: intel-dpdk-node-policy
  namespace: openshift-sriov-network-operator
spec:
  resourceName: intelnics
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  priority: <priority>
  numVfs: <num>
  nicSelector:
    vendor: "8086"
    deviceID: "158b"
    pfNames: ["<pf_name>", ...]
    rootDevices: ["<pci_bus_id>", "..."]
  deviceType: vfio-pci 1
```

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

> **注意**
>
> 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** object, the SR-IOV Operator may drain the nodes, and in some cases, reboot nodes. It may take several minutes for a configuration change to apply. Ensure that there are enough available nodes in your cluster to handle the evicted workload beforehand.
>
> After the configuration update is applied, all the pods in **openshift-sriov-network-operator** namespace will change to a **Running** status.

2. Create the **SriovNetworkNodePolicy** object by running the following command:

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

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

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

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

注意

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

4. Create the **SriovNetworkNodePolicy** object by running the following command:

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

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

```
apiVersion: v1
kind: Pod
metadata:
  name: dpdk-app
  namespace: <target_namespace> 1
  annotations:
    k8s.v1.cni.cncf.io/networks: intel-dpdk-network
spec:
  containers:
  - name: testpmd
    image: <DPDK_image> 2
    securityContext:
     capabilities:
        add: ["IPC_LOCK"] 3
    volumeMounts:
    - mountPath: /dev/hugepages 4
      name: hugepage
    resources:
      limits:
        openshift.io/intelnics: "1" 5
        memory: "1Gi"
        cpu: "4" 6
        hugepages-1Gi: "4Gi" 7
      requests:
        openshift.io/intelnics: "1"
        memory: "1Gi"
        cpu: "4"
        hugepages-1Gi: "4Gi"
    command: ["sleep", "infinity"]
  volumes:
  - name: hugepage
    emptyDir:
      medium: HugePages
```

1. Specify the same **target_namespace** where the **SriovNetwork** object **intel-dpdk-network** is created. If you would like to create the pod in a different namespace, change **target_namespace** in both the **Pod** spec and the **SriovNetowrk** object.

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

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

4. Mount a hugepage volume to the DPDK pod under **/dev/hugepages**. The hugepage volume is backed by the emptyDir volume type with the medium being **Hugepages**.

5. Optional: Specify the number of DPDK devices allocated to DPDK pod. This resource request and limit, if not explicitly specified, will be automatically added by the SR-IOV network resource injector. The SR-IOV network resource injector is an admission controller component managed by the SR-IOV Operator. It is enabled by default and can be disabled by setting **enableInjector** option to **false** in the default **SriovOperatorConfig** CR.

6. Specify the number of CPUs. The DPDK pod usually requires exclusive CPUs to be allocated from the kubelet. This is achieved by setting CPU Manager policy to **static** and creating a pod with **Guaranteed** QoS.

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

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

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

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

Procedure

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

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

❶ Specify the device hex code of the 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**. 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.

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

注意

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

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

2. Create the **SriovNetworkNodePolicy** object by running the following command:

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

3. Create the following **SriovNetwork** object, 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: |- ❶
  ...
vlan: <vlan>
resourceName: mlxnics
```

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.

注意

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

4. Create the **SriovNetworkNodePolicy** object 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> 2
    securityContext:
     capabilities:
       add: ["IPC_LOCK","NET_RAW"] 3
    volumeMounts:
    - mountPath: /dev/hugepages 4
      name: hugepage
    resources:
      limits:
        openshift.io/mlxnics: "1" 5
        memory: "1Gi"
        cpu: "4" 6
        hugepages-1Gi: "4Gi" 7
      requests:
        openshift.io/mlxnics: "1"
        memory: "1Gi"
        cpu: "4"
        hugepages-1Gi: "4Gi"
    command: ["sleep", "infinity"]
  volumes:
  - name: hugepage
    emptyDir:
      medium: HugePages
```

1. Specify the same **target_namespace** where **SriovNetwork** object **mlx-dpdk-network** is created. If you would like to create the pod in a different namespace, change

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

3. Specify the **IPC_LOCK** capability which is required by the application to allocate hugepage memory inside the container and **NET_RAW** for the application to access the network interface.

4. Mount the hugepage volume to the DPDK pod under **/dev/hugepages**. The hugepage volume is backed by the emptyDir volume type with the medium being **Hugepages**.

5. Optional: Specify the number of DPDK devices allocated to the DPDK pod. This resource request and limit, if not explicitly specified, will be automatically added by SR-IOV network resource injector. The SR-IOV network resource injector is an admission controller component managed by SR-IOV Operator. It is enabled by default and can be disabled by setting the **enableInjector** option to **false** in the default **SriovOperatorConfig** CR.

6. Specify the number of CPUs. The DPDK pod usually requires exclusive CPUs be allocated from kubelet. This is achieved by setting CPU Manager policy to **static** and creating a pod with **Guaranteed** QoS.

7. Specify hugepage size **hugepages-1Gi** or **hugepages-2Mi** and the quantity of hugepages that will be allocated to DPDK pod. Configure **2Mi** and **1Gi** hugepages separately. Configuring **1Gi** hugepage requires adding kernel arguments to Nodes.

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

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

11.8.5. 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** object, and then save the YAML in the **mlx-rdma-node-policy.yaml** file.

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: mlx-rdma-node-policy
  namespace: openshift-sriov-network-operator
spec:
  resourceName: mlxnics
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  priority: <priority>
  numVfs: <num>
  nicSelector:
    vendor: "15b3"
    deviceID: "1015" 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.

注意

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

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

2. Create the **SriovNetworkNodePolicy** object by running the following command:

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

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

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

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.

注意

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

4. Create the **SriovNetworkNodePolicy** object by running the following command:

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

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

```
apiVersion: v1
kind: Pod
metadata:
  name: rdma-app
  namespace: <target_namespace> 1
  annotations:
    k8s.v1.cni.cncf.io/networks: mlx-rdma-network
spec:
  containers:
  - name: testpmd
    image: <RDMA_image> 2
    securityContext:
     capabilities:
        add: ["IPC_LOCK"] 3
    volumeMounts:
    - mountPath: /dev/hugepages 4
      name: hugepage
    resources:
      limits:
        memory: "1Gi"
        cpu: "4" 5
        hugepages-1Gi: "4Gi" 6
      requests:
        memory: "1Gi"
        cpu: "4"
        hugepages-1Gi: "4Gi"
    command: ["sleep", "infinity"]
  volumes:
  - name: hugepage
    emptyDir:
      medium: HugePages
```

1 Specify the same **target_namespace** where **SriovNetwork** object **mlx-rdma-network** is created. If you would like to create the pod in a different namespace, change **target_namespace** in both **Pod** spec and **SriovNetowrk** object.

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
```

第 12 章 OPENSHIFT SDN DEFAULT CNI NETWORK PROVIDER

12.1. ABOUT THE OPENSHIFT SDN DEFAULT CNI NETWORK PROVIDER

OpenShift Container Platform uses a software-defined networking (SDN) approach to provide a unified cluster network that enables communication between pods across the OpenShift Container Platform cluster. This pod network is established and maintained by the OpenShift SDN, which configures an overlay network using Open vSwitch (OVS).

12.1.1. OpenShift SDN network isolation modes

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

- *Network policy* mode allows project administrators to configure their own isolation policies using **NetworkPolicy** objects. Network policy is the default mode in OpenShift Container Platform 4.5.

- *Multitenant* mode provides project-level isolation for pods and services. Pods from different projects cannot send packets to or receive packets from pods and services of a different project. You can disable isolation for a project, allowing it to send network traffic to all pods and services in the entire cluster and receive network traffic from those pods and services.

- *Subnet* mode provides a flat pod network where every pod can communicate with every other pod and service. The network policy mode provides the same functionality as subnet mode.

12.1.2. Supported default CNI network provider feature matrix

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

表 12.1. Default CNI network provider feature comparison

Feature	OpenShift SDN	OVN-Kubernetes [1]
Egress IPs	Supported	Not supported
Egress firewall [2]	Supported	Not supported
Egress router	Supported	Not supported
Kubernetes network policy	Partially supported [3]	Supported
Multicast	Supported	Supported

1. Available only as a Technology Preview feature in OpenShift Container Platform 4.5.

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

3. Does not support egress rules and some **ipBlock** rules.

12.2. CONFIGURING EGRESS IPS FOR A PROJECT

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

12.2.1. 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.

重要

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** object. 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** object, but no node hosts that egress IP address, then egress traffic from the namespace will be dropped.

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

重要

The following limitations apply when using egress IP addresses with the OpenShift SDN cluster network provider:

- You cannot use manually assigned and automatically assigned egress IP addresses on the same nodes.
- If you manually assign egress IP addresses from an IP address range, you must not make that range available for automatic IP assignment.
- You cannot share egress IP addresses across multiple namespaces using the OpenShift SDN egress IP address implementation.

注意

If you use OpenShift SDN in multitenant mode, you cannot use egress IP addresses with any namespace that is joined to another namespace by the projects that are associated with them. For example, if **project1** and **project2** are joined by running the **oc adm pod-network join-projects --to=project1 project2** command, neither project can use an egress IP address. For more information, see BZ#1645577.

12.2.1.1. 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.

12.2.1.2. Considerations when using manually assigned egress IP addresses

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

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.

12.2.2. 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

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

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

Procedure

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

```
$ oc patch netnamespace <project_name> --type=merge -p \ 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"]}'
```

注意

Because OpenShift SDN manages the **NetNamespace** object, you can make changes only by modifying the existing **NetNamespace** object. Do not create a new **NetNamespace** object.

2. Indicate which nodes can host egress IP addresses by setting the **egressCIDRs** parameter for each host using the following JSON:

```
$ oc patch hostsubnet <node_name> --type=merge -p \ 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

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.

12.2.3. 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

- You have access to the cluster as a user with the **cluster-admin** role.
- You have installed the OpenShift CLI (**oc**).

Procedure

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

```
$ oc patch netnamespace <project> --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.

注意

Because OpenShift SDN manages the **NetNamespace** object, you can make changes only by modifying the existing **NetNamespace** object. Do not create a new **NetNamespace** object.

2. Manually assign the egress IP 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 node.

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.

12.3. 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.

12.3.1. 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.

重要

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.

⚠ 警告

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.

12.3.1.1. Limitations of an egress firewall

An egress firewall has the following limitations:

- No project can have more than one EgressNetworkPolicy object.
- A maximum of 1 EgressNetworkPolicy object with a maximum of 50 rules can be defined per project.
- The **default** project cannot use egress network policy.
- When using the OpenShift SDN default Container Network Interface (CNI) network provider in multitenant mode, the following limitations apply:
    - Global projects cannot use an egress firewall. You can make a project global by using the **oc adm pod-network make-projects-global** command.
    - Projects merged by using the **oc adm pod-network join-projects** command cannot use an egress firewall in any of the joined projects.

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

12.3.1.2. 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.

12.3.1.3. 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.

注意

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.

12.3.2. EgressNetworkPolicy custom resource (CR) object

The following YAML describes an EgressNetworkPolicy CR object:

```
apiVersion: network.openshift.io/v1
kind: EgressNetworkPolicy
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.

12.3.2.1. EgressNetworkPolicy rules

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

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

1 Specify the type of rule. The value must be either **Allow** or **Deny**.

2 Specify a value for either the **cidrSelector** key or the **dnsName** key for the rule. You cannot use both keys in a rule.

3 Specify an IP address range in CIDR format.

4 Specify a domain name.

12.3.2.2. Example EgressNetworkPolicy CR object

The following example defines several egress firewall policy rules:

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

```
kind: EgressNetworkPolicy
metadata:
  name: default-rules 1
spec:
  egress: 2
  - type: Allow
    to:
      cidrSelector: 1.2.3.0/24
  - type: Allow
    to:
      dnsName: www.example.com
  - type: Deny
    to:
      cidrSelector: 0.0.0.0/0
```

1 The name for the policy object.

2 A collection of egress firewall policy rule objects.

12.3.3. Creating an egress firewall policy object

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

重要

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

Prerequisites

- A cluster that uses the OpenShift SDN default Container Network Interface (CNI) network provider plug-in.
- Install the OpenShift CLI (**oc**).
- You must log in to the cluster as a cluster administrator.

Procedure

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

```
$ 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
```

Example output

```
egressnetworkpolicy.network.openshift.io/default-rules created
```

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

12.4. EDITING AN EGRESS FIREWALL FOR A PROJECT

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

12.4.1. 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 CLI (**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. Optional: If you did not save a copy of the EgressNetworkPolicy object when you created the egress network firewall, enter the following command to create a copy.

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

12.4.2. EgressNetworkPolicy custom resource (CR) object

The following YAML describes an EgressNetworkPolicy CR object:

```
apiVersion: network.openshift.io/v1
kind: EgressNetworkPolicy
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.

12.4.2.1. 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> ①
  to: ②
    cidrSelector: <cidr> ③
    dnsName: <dns-name> ④
```

① Specify the type of rule. The value must be either **Allow** or **Deny**.

② Specify a value for either the **cidrSelector** key or the **dnsName** key for the rule. You cannot use both keys in a rule.

③ Specify an IP address range in CIDR format.

④ Specify a domain name.

12.4.2.2. Example EgressNetworkPolicy CR object

The following example defines several egress firewall policy rules:

```
apiVersion: network.openshift.io/v1
kind: EgressNetworkPolicy
metadata:
  name: default-rules ①
spec:
  egress: ②
  - type: Allow
    to:
      cidrSelector: 1.2.3.0/24
  - type: Allow
    to:
      dnsName: www.example.com
  - type: Deny
    to:
      cidrSelector: 0.0.0.0/0
```

1 The name for the policy object.

2 A collection of egress firewall policy rule objects.

12.5. 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.

12.5.1. 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 CLI (**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>
```

12.6. CONSIDERATIONS FOR THE USE OF AN EGRESS ROUTER POD

12.6.1. About an egress router pod

The OpenShift Container Platform egress router pod redirects traffic to a specified remote server, using a private source IP address that is not used for any other purpose. This allows you to send network traffic to servers that are set up to allow access only from specific IP addresses.

注意

The egress router pod is not intended for every outgoing connection. Creating large numbers of egress router pods can exceed the limits of your network hardware. For example, creating an egress router pod for every project or application could exceed the number of local MAC addresses that the network interface can handle before reverting to filtering MAC addresses in software.

重要

The egress router image is not compatible with Amazon AWS, Azure Cloud, or any other cloud platform that does not support layer 2 manipulations due to their incompatibility with macvlan traffic.

12.6.1.1. Egress router modes

In *redirect mode*, an egress router pod sets up iptables rules to redirect traffic from its own IP address to one or more destination IP addresses. Client pods that need to use the reserved source IP address must be modified to connect to the egress router rather than connecting directly to the destination IP.

In *HTTP proxy mode*, an egress router pod runs as an HTTP proxy on port **8080**. This mode only works for clients that are connecting to HTTP-based or HTTPS-based services, but usually requires fewer changes to the client pods to get them to work. Many programs can be told to use an HTTP proxy by setting an environment variable.

In *DNS proxy mode*, an egress router pod runs as a DNS proxy for TCP-based services from its own IP address to one or more destination IP addresses. To make use of the reserved, source IP address, client pods must be modified to connect to the egress router pod rather than connecting directly to the destination IP address. This modification ensures that external destinations treat traffic as though it were coming from a known source.

Redirect mode works for all services except for HTTP and HTTPS. For HTTP and HTTPS services, use HTTP proxy mode. For TCP-based services with IP addresses or domain names, use DNS proxy mode.

12.6.1.2. Egress router pod implementation

The egress router pod setup is performed by an initialization container. That container runs in a privileged context so that it can configure the macvlan interface and set up **iptables** rules. After the initialization container finishes setting up the **iptables** rules, it exits. Next the egress router pod executes the container to handle the egress router traffic. The image used varies depending on the egress router mode.

The environment variables determine which addresses the egress-router image uses. The image configures the macvlan interface to use **EGRESS_SOURCE** as its IP address, with **EGRESS_GATEWAY** as the IP address for the gateway.

Network Address Translation (NAT) rules are set up so that connections to the cluster IP address of the pod on any TCP or UDP port are redirected to the same port on IP address specified by the **EGRESS_DESTINATION** variable.

If only some of the nodes in your cluster are capable of claiming the specified source IP address and using the specified gateway, you can specify a **nodeName** or **nodeSelector** indicating which nodes are acceptable.

12.6.1.3. Deployment considerations

An egress router pod adds an additional IP address and MAC address to the primary network interface of the node. As a result, you might need to configure your hypervisor or cloud provider to allow the additional address.

Red Hat OpenStack Platform (RHOSP)

If you are deploying OpenShift Container Platform on RHOSP, you must whitelist the IP and MAC addresses on your OpenStack environment, otherwise communication will fail:

-

```
$ openstack port set --allowed-address \
  ip_address=<ip_address>,mac_address=<mac_address> <neutron_port_uuid>
```

Red Hat Virtualization (RHV)

If you are using RHV, you must select **No Network Filter** for the Virtual Network Interface Card (vNIC).

VMware vSphere

If you are using VMware vSphere, see the VMWare documentation for securing vSphere standard switches. View and change VMWare vSphere default settings by selecting the host virtual switch from the vSphere Web Client.

Specifically, ensure that the following are enabled:

- MAC Address Changes
- Forged Transits
- Promiscuous Mode Operation

12.6.1.4. Failover configuration

To avoid downtime, you can deploy an egress router pod with a **Deployment** resource, as in the following example. To create a new **Service** object for the example deployment, use the **oc expose deployment/egress-demo-controller** command.

```
apiVersion: v1
kind: Deployment
metadata:
  name: egress-demo-controller
spec:
  replicas: 1 ❶
  selector:
    name: egress-router
  template:
    metadata:
      name: egress-router
      labels:
        name: egress-router
      annotations:
        pod.network.openshift.io/assign-macvlan: "true"
    spec: ❷
      initContainers:
        ...
      containers:
        ...
```

❶ Ensure that replicas is set to **1**, because only one pod can use a given egress source IP address at any time. This means that only a single copy of the router runs on a node.

❷ Specify the **Pod** object template for the egress router pod.

12.6.2. Additional resources

- Deploying an egress router in redirection mode
- Deploying an egress router in HTTP proxy mode
- Deploying an egress router in DNS proxy mode

12.7. DEPLOYING AN EGRESS ROUTER POD IN REDIRECT MODE

As a cluster administrator, you can deploy an egress router pod that is configured to redirect traffic to specified destination IP addresses.

12.7.1. Egress router pod specification for redirect mode

Define the configuration for an egress router pod in the **Pod** object. The following YAML describes the fields for the configuration of an egress router pod in redirect mode:

```
apiVersion: v1
kind: Pod
metadata:
  name: egress-1
  labels:
    name: egress-1
  annotations:
    pod.network.openshift.io/assign-macvlan: "true" 1
spec:
  initContainers:
  - name: egress-router
    image: registry.redhat.io/openshift4/ose-egress-router
    securityContext:
      privileged: true
    env:
    - name: EGRESS_SOURCE 2
      value: <egress_router>
    - name: EGRESS_GATEWAY 3
      value: <egress_gateway>
    - name: EGRESS_DESTINATION 4
      value: <egress_destination>
    - name: EGRESS_ROUTER_MODE
      value: init
  containers:
  - name: egress-router-wait
    image: registry.redhat.io/openshift3/ose-pod
```

1 Before starting the **egress-router** container, create a macvlan network interface on the primary network interface and move that interface into the pod network namespace. You must include the quotation marks around the **"true"** value. To create the macvlan interface on a network interface other than the primary one, set the annotation value to the name of that interface. For example, **eth1**.

2 IP address from the physical network that the node is on that is reserved for use by the egress router pod. Optional: You can include the subnet length, the **/24** suffix, so that a proper route to the local subnet is set. If you do not specify a subnet length, then the egress router can access only the host specified with the **EGRESS_GATEWAY** variable and no other hosts on the subnet.

3 Same value as the default gateway used by the node.

4 External server to direct traffic to. Using this example, connections to the pod are redirected to **203.0.113.25**, with a source IP address of **192.168.12.99**.

Example egress router Pod specification

```
apiVersion: v1
kind: Pod
metadata:
  name: egress-multi
  labels:
    name: egress-multi
  annotations:
    pod.network.openshift.io/assign-macvlan: "true"
spec:
  initContainers:
  - name: egress-router
    image: registry.redhat.io/openshift4/ose-egress-router
    securityContext:
      privileged: true
    env:
    - name: EGRESS_SOURCE
      value: 192.168.12.99/24
    - name: EGRESS_GATEWAY
      value: 192.168.12.1
    - name: EGRESS_DESTINATION
      value: |
        80   tcp 203.0.113.25
        8080 tcp 203.0.113.26 80
        8443 tcp 203.0.113.26 443
        203.0.113.27
    - name: EGRESS_ROUTER_MODE
      value: init
  containers:
  - name: egress-router-wait
    image: registry.redhat.io/openshift3/ose-pod
```

12.7.2. Egress destination configuration format

When an egress router pod is deployed in redirect mode, you can specify redirection rules by using one or more of the following formats:

- **<port> <protocol> <ip_address>** - Incoming connections to the given **<port>** should be redirected to the same port on the given **<ip_address>**. **<protocol>** is either **tcp** or **udp**.
- **<port> <protocol> <ip_address> <remote_port>** - As above, except that the connection is redirected to a different **<remote_port>** on **<ip_address>**.
- **<ip_address>** - If the last line is a single IP address, then any connections on any other port will be redirected to the corresponding port on that IP address. If there is no fallback IP address then connections on other ports are rejected.

In the example that follows several rules are defined:

- The first line redirects traffic from local port **80** to port **80** on **203.0.113.25**.
- The second and third lines redirect local ports **8080** and **8443** to remote ports **80** and **443** on **203.0.113.26**.
- The last line matches traffic for any ports not specified in the previous rules.

Example configuration

```
80   tcp 203.0.113.25
8080 tcp 203.0.113.26 80
8443 tcp 203.0.113.26 443
203.0.113.27
```

12.7.3. Deploying an egress router pod in redirect mode

In *redirect mode*, an egress router pod sets up iptables rules to redirect traffic from its own IP address to one or more destination IP addresses. Client pods that need to use the reserved source IP address must be modified to connect to the egress router rather than connecting directly to the destination IP.

Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

Procedure

1. Create an egress router pod.
2. To ensure that other pods can find the IP address of the egress router pod, create a service to point to the egress router pod, as in the following example:

```
apiVersion: v1
kind: Service
metadata:
  name: egress-1
spec:
  ports:
  - name: http
    port: 80
  - name: https
    port: 443
  type: ClusterIP
  selector:
    name: egress-1
```

Your pods can now connect to this service. Their connections are redirected to the corresponding ports on the external server, using the reserved egress IP address.

12.7.4. Additional resources

- Configuring an egress router destination mappings with a ConfigMap

12.8. DEPLOYING AN EGRESS ROUTER POD IN HTTP PROXY MODE

As a cluster administrator, you can deploy an egress router pod configured to proxy traffic to specified HTTP and HTTPS-based services.

12.8.1. Egress router pod specification for HTTP mode

Define the configuration for an egress router pod in the **Pod** object. The following YAML describes the fields for the configuration of an egress router pod in HTTP mode:

```
apiVersion: v1
kind: Pod
metadata:
  name: egress-1
  labels:
    name: egress-1
  annotations:
    pod.network.openshift.io/assign-macvlan: "true" 1
spec:
  initContainers:
  - name: egress-router
    image: registry.redhat.io/openshift4/ose-egress-router
    securityContext:
      privileged: true
    env:
    - name: EGRESS_SOURCE 2
      value: <egress-router>
    - name: EGRESS_GATEWAY 3
      value: <egress-gateway>
    - name: EGRESS_ROUTER_MODE
      value: http-proxy
  containers:
  - name: egress-router-pod
    image: registry.redhat.io/ose-egress-http-proxy
    env:
    - name: EGRESS_HTTP_PROXY_DESTINATION 4
      value: |-
        ...
    ...
```

1 Before starting the **egress-router** container, create a macvlan network interface on the primary network interface and move that interface into the pod network namespace. You must include the quotation marks around the **"true"** value. To create the macvlan interface on a network interface other than the primary one, set the annotation value to the name of that interface. For example, **eth1**.

2 IP address from the physical network that the node is on that is reserved for use by the egress router pod. Optional: You can include the subnet length, the **/24** suffix, so that a proper route to the local subnet is set. If you do not specify a subnet length, then the egress router can access only the host specified with the **EGRESS_GATEWAY** variable and no other hosts on the subnet.

3 Same value as the default gateway used by the node.

4 A string or YAML multi-line string specifying how to configure the proxy. Note that this is specified as an environment variable in the HTTP proxy container, not with the other environment variables

in the init container.

12.8.2. Egress destination configuration format

When an egress router pod is deployed in HTTP proxy mode, you can specify redirection rules by using one or more of the following formats. Each line in the configuration specifies one group of connections to allow or deny:

- An IP address allows connections to that IP address, such as **192.168.1.1**.
- A CIDR range allows connections to that CIDR range, such as **192.168.1.0/24**.
- A host name allows proxying to that host, such as **www.example.com**.
- A domain name preceded by ***.** allows proxying to that domain and all of its subdomains, such as ***.example.com**.
- A **!** followed by any of the previous match expressions denies the connection instead.
- If the last line is *****, then anything that is not explicitly denied is allowed. Otherwise, anything that is not allowed is denied.

You can also use ***** to allow connections to all remote destinations.

Example configuration

```
!*.example.com
!192.168.1.0/24
192.168.2.1
*
```

12.8.3. Deploying an egress router pod in HTTP proxy mode

In *HTTP proxy mode*, an egress router pod runs as an HTTP proxy on port **8080**. This mode only works for clients that are connecting to HTTP-based or HTTPS-based services, but usually requires fewer changes to the client pods to get them to work. Many programs can be told to use an HTTP proxy by setting an environment variable.

Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

Procedure

1. Create an egress router pod.
2. To ensure that other pods can find the IP address of the egress router pod, create a service to point to the egress router pod, as in the following example:

```
apiVersion: v1
kind: Service
metadata:
  name: egress-1
spec:
  ports:
  - name: http-proxy
    port: 8080 ①
  type: ClusterIP
  selector:
    name: egress-1
```

① Ensure the **http** port is set to **8080**.

3. To configure the client pod (not the egress proxy pod) to use the HTTP proxy, set the **http_proxy** or **https_proxy** variables:

```
apiVersion: v1
kind: Pod
metadata:
  name: app-1
  labels:
    name: app-1
spec:
  containers:
    env:
    - name: http_proxy
      value: http://egress-1:8080/ ①
    - name: https_proxy
      value: http://egress-1:8080/
    ...
```

① The service created in the previous step.

注意

Using the **http_proxy** and **https_proxy** environment variables is not necessary for all setups. If the above does not create a working setup, then consult the documentation for the tool or software you are running in the pod.

12.8.4. Additional resources

- Configuring an egress router destination mappings with a ConfigMap

12.9. DEPLOYING AN EGRESS ROUTER POD IN DNS PROXY MODE

As a cluster administrator, you can deploy an egress router pod configured to proxy traffic to specified DNS names and IP addresses.

12.9.1. Egress router pod specification for DNS mode

Define the configuration for an egress router pod in the **Pod** object. The following YAML describes the fields for the configuration of an egress router pod in DNS mode:

```
apiVersion: v1
```

```
kind: Pod
metadata:
  name: egress-1
  labels:
    name: egress-1
  annotations:
    pod.network.openshift.io/assign-macvlan: "true" ①
spec:
  initContainers:
  - name: egress-router
    image: registry.redhat.io/openshift4/ose-egress-router
    securityContext:
      privileged: true
    env:
    - name: EGRESS_SOURCE ②
      value: <egress-router>
    - name: EGRESS_GATEWAY ③
      value: <egress-gateway>
    - name: EGRESS_ROUTER_MODE
      value: dns-proxy
  containers:
  - name: egress-router-pod
    image: registry.redhat.io/openshift4/ose-egress-dns-proxy
    securityContext:
      privileged: true
    env:
    - name: EGRESS_DNS_PROXY_DESTINATION ④
      value: |-
        ...
    - name: EGRESS_DNS_PROXY_DEBUG ⑤
      value: "1"
    ...
```

① Before starting the **egress-router** container, create a macvlan network interface on the primary network interface and move that interface into the pod network namespace. You must include the quotation marks around the **"true"** value. To create the macvlan interface on a network interface other than the primary one, set the annotation value to the name of that interface. For example, **eth1**.

② IP address from the physical network that the node is on that is reserved for use by the egress router pod. Optional: You can include the subnet length, the **/24** suffix, so that a proper route to the local subnet is set. If you do not specify a subnet length, then the egress router can access only the host specified with the **EGRESS_GATEWAY** variable and no other hosts on the subnet.

③ Same value as the default gateway used by the node.

④ Specify a list of one or more proxy destinations.

⑤ Optional: Specify to output the DNS proxy log output to **stdout**.

12.9.2. Egress destination configuration format

When the router is deployed in DNS proxy mode, you specify a list of port and destination mappings. A destination may be either an IP address or a DNS name.

An egress router pod supports the following formats for specifying port and destination mappings:

Port and remote address

You can specify a source port and a destination host by using the two field format: **<port> <remote_address>**.

The host can be an IP address or a DNS name. If a DNS name is provided, DNS resolution occurs at runtime. For a given host, the proxy connects to the specified source port on the destination host when connecting to the destination host IP address.

Port and remote address pair example

```
80 172.16.12.11
100 example.com
```

Port, remote address, and remote port

You can specify a source port, a destination host, and a destination port by using the three field format: **<port> <remote_address> <remote_port>**.

The three field format behaves identically to the two field version, with the exception that the destination port can be different than the source port.

Port, remote address, and remote port example

```
8080 192.168.60.252 80
8443 web.example.com 443
```

12.9.3. Deploying an egress router pod in DNS proxy mode

In *DNS proxy mode* , an egress router pod acts as a DNS proxy for TCP-based services from its own IP address to one or more destination IP addresses.

Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

Procedure

1. Create an egress router pod.
2. Create a service for the egress router pod:
   a. Create a file named **egress-router-service.yaml** that contains the following YAML. Set **spec.ports** to the list of ports that you defined previously for the **EGRESS_DNS_PROXY_DESTINATION** environment variable.

```
apiVersion: v1
kind: Service
metadata:
  name: egress-dns-svc
spec:
  ports:
    ...
  type: ClusterIP
  selector:
    name: egress-dns-proxy
```

For example:

```
apiVersion: v1
kind: Service
metadata:
  name: egress-dns-svc
spec:
  ports:
  - name: con1
    protocol: TCP
    port: 80
    targetPort: 80
  - name: con2
    protocol: TCP
    port: 100
    targetPort: 100
  type: ClusterIP
  selector:
    name: egress-dns-proxy
```

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

```
$ oc create -f egress-router-service.yaml
```

Pods can now connect to this service. The connections are proxied to the corresponding ports on the external server, using the reserved egress IP address.

12.9.4. Additional resources

- Configuring an egress router destination mappings with a ConfigMap

12.10. CONFIGURING AN EGRESS ROUTER POD DESTINATION LIST FROM A CONFIG MAP

As a cluster administrator, you can define a **ConfigMap** object that specifies destination mappings for an egress router pod. The specific format of the configuration depends on the type of egress router pod. For details on the format, refer to the documentation for the specific egress router pod.

12.10.1. Configuring an egress router destination mappings with a config map

For a large or frequently-changing set of destination mappings, you can use a config map to externally maintain the list. An advantage of this approach is that permission to edit the config map can be delegated to users without **cluster-admin** privileges. Because the egress router pod requires a privileged container, it is not possible for users without **cluster-admin** privileges to edit the pod definition directly.

注意

The egress router pod does not automatically update when the config map changes. You must restart the egress router pod to get updates.

Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

Procedure

1. Create a file containing the mapping data for the egress router pod, as in the following example:

```
# Egress routes for Project "Test", version 3

80   tcp 203.0.113.25

8080 tcp 203.0.113.26 80
8443 tcp 203.0.113.26 443

# Fallback
203.0.113.27
```

You can put blank lines and comments into this file.

2. Create a **ConfigMap** object from the file:

```
$ oc delete configmap egress-routes --ignore-not-found
```

```
$ oc create configmap egress-routes \
  --from-file=destination=my-egress-destination.txt
```

In the previous command, the **egress-routes** value is the name of the **ConfigMap** object to create and **my-egress-destination.txt** is the name of the file that the data is read from.

3. Create an egress router pod definition and specify the **configMapKeyRef** stanza for the **EGRESS_DESTINATION** field in the environment stanza:

```
...
env:
- name: EGRESS_DESTINATION
  valueFrom:
    configMapKeyRef:
      name: egress-routes
      key: destination
...
```

12.10.2. Additional resources

- Redirect mode
- HTTP proxy mode

- DNS proxy mode

12.11. ENABLING MULTICAST FOR A PROJECT

12.11.1. About multicast

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

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

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

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

- Multicast packets sent by a pod will be delivered to all other pods in the project, regardless of **NetworkPolicy** objects. Pods might be able to communicate over multicast even when they cannot communicate over unicast.
- Multicast packets sent by a pod in one project will never be delivered to pods in any other project, even if there are **NetworkPolicy** objects that allow communication between the projects.

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

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

12.11.2. Enabling multicast between pods

You can enable multicast between pods for your project.

Prerequisites

- Install the OpenShift CLI (**oc**).
- You must log in to the cluster with a user that has the **cluster-admin** role.

Procedure

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

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

Verification

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

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

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

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

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

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

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

4. Start the multicast listener.

   a. Get the IP address for the Pod:

      ```
      $ POD_IP=$(oc get pods mlistener -o jsonpath='{.status.podIP}')
      ```

   b. To start the multicast listener, in a new terminal window or tab, enter the following command:

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

5. Start the multicast transmitter.

   a. Get the pod network IP address range:

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

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

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

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

   ```
   mlistener
   ```

12.12. DISABLING MULTICAST FOR A PROJECT

12.12.1. Disabling multicast between pods

You can disable multicast between pods for your project.

Prerequisites

- Install the OpenShift CLI (**oc**).
- You must log in to the cluster with a user that has the **cluster-admin** role.

Procedure

- Disable multicast by running the following command:

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

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

12.13. CONFIGURING NETWORK ISOLATION USING OPENSHIFT SDN

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

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

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

12.13.1. Prerequisites

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

12.13.2. Joining projects

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

Prerequisites

- Install the OpenShift CLI (**oc**).
- You must log in to the cluster with a user that has the **cluster-admin** role.

Procedure

1. Use the following command to join projects to an existing project network:

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

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

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

   ```
   $ oc get netnamespaces
   ```

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

12.13.3. Isolating a project

You can isolate a project so that pods and services in other projects cannot access its pods and services.

Prerequisites

- Install the OpenShift CLI (**oc**).
- You must log in to the cluster with a user that has the **cluster-admin** role.

Procedure

- To isolate the projects in the cluster, run the following command:

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

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

12.13.4. Disabling network isolation for a project

You can disable network isolation for a project.

Prerequisites

- Install the OpenShift CLI (**oc**).
- You must log in to the cluster with a user that has the **cluster-admin** role.

Procedure

- Run the following command for the project:

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

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

12.14. 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.

12.14.1. 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.

12.14.2. kube-proxy configuration parameters

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

重要

Because of performance improvements introduced in OpenShift Container Platform 4.3 and greater, adjusting the **iptablesSyncPeriod** parameter is no longer necessary.

表 12.2. Parameters

Parameter	Description	Values	Default
iptablesSyncPeriod	The refresh period for **iptables** rules.	A time interval, such as **30s** or **2m**. Valid suffixes include **s**, **m**, and **h** and are described in the Go time package documentation.	**30s**
proxyArguments.iptables-min-sync-period	The minimum duration before refreshing **iptables** rules. This parameter ensures that the refresh does not happen too frequently. By default, a refresh starts as soon as a change that affects **iptables** rules occurs.	A time interval, such as **30s** or **2m**. Valid suffixes include **s**, **m**, and **h** and are described in the Go time package	**0s**

12.14.3. Modifying the kube-proxy configuration

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

Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in to a running cluster with the **cluster-admin** role.

Procedure

1. Edit the **Network.operator.openshift.io** custom resource (CR) by running the following command:

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

2. Modify the **kubeProxyConfig** parameter in the CR with your changes to the kube-proxy configuration, such as in the following example CR:

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  kubeProxyConfig:
    iptablesSyncPeriod: 30s
    proxyArguments:
      iptables-min-sync-period: ["30s"]
```

3. Save the file and exit the text editor.
The syntax is validated by the **oc** command when you save the file and exit the editor. If your modifications contain a syntax error, the editor opens the file and displays an error message.

4. Enter the following command to confirm the configuration update:

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

Example output

```
apiVersion: v1
items:
- apiVersion: operator.openshift.io/v1
  kind: Network
  metadata:
    name: cluster
  spec:
    clusterNetwork:
    - cidr: 10.128.0.0/14
      hostPrefix: 23
    defaultNetwork:
      type: OpenShiftSDN
    kubeProxyConfig:
      iptablesSyncPeriod: 30s
      proxyArguments:
        iptables-min-sync-period:
        - 30s
    serviceNetwork:
    - 172.30.0.0/16
  status: {}
kind: List
```

5. Optional: Enter the following command to confirm that the Cluster Network Operator accepted the configuration change:

```
$ oc get clusteroperator network
```

Example output

```
NAME      VERSION     AVAILABLE   PROGRESSING   DEGRADED   SINCE
network   4.1.0-0.9   True        False         False      1m
```

The **AVAILABLE** field is **True** when the configuration update is applied successfully.

第 13 章 OVN-KUBERNETES DEFAULT CNI NETWORK PROVIDER

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

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

13.1.1. OVN-Kubernetes features

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

- Uses OVN (Open Virtual Network) to manage network traffic flows. OVN is a community developed, vendor agnostic network virtualization solution.
- Implements Kubernetes network policy support, including ingress and egress rules.
- Uses the Geneve (Generic Network Virtualization Encapsulation) protocol rather than VXLAN to create an overlay network between nodes.

13.1.2. Supported default CNI network provider feature matrix

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

表 13.1. Default CNI network provider feature comparison

Feature	OVN-Kubernetes [1]	OpenShift SDN
Egress IPs	Not supported	Supported
Egress firewall [2]	Not supported	Supported
Egress router	Not supported	Supported
Kubernetes network policy	Supported	Partially supported [3]
Multicast	Supported	Supported

1. Available only as a Technology Preview feature in OpenShift Container Platform 4.5.
2. Egress firewall is also known as egress network policy in OpenShift SDN. This is not the same as network policy egress.
3. Does not support egress rules and some **ipBlock** rules.

13.1.3. Exposed metrics for OVN-Kubernetes

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

表 13.2. Metrics exposed by OVN-Kubernetes

Name	Description
ovnkube_master_pod_creation_latency_seconds	The latency between when a pod is created and when the pod is annotated by OVN-Kubernetes. The higher the latency, the more time that elapses before a pod is available for network connectivity.

Additional resources

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

13.2. MIGRATE FROM THE OPENSHIFT SDN DEFAULT CNI NETWORK PROVIDER

As a cluster administrator, you can migrate to the OVN-Kubernetes default Container Network Interface (CNI) network provider from the OpenShift SDN default CNI network provider.

重要

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.

To learn more about OVN-Kubernetes, read About the OVN-Kubernetes network provider .

13.2.1. Migrating to the OVN-Kubernetes default CNI network provider

As a cluster administrator, you can change the default Container Network Interface (CNI) network provider for your cluster to OVN-Kubernetes. During the migration, you must reboot every node in your cluster.

重要

While performing the migration, your cluster is unavailable and workloads might be interrupted. Perform the migration only when an interruption in service is acceptable.

Prerequisites

- Install the OpenShift CLI (**oc**).
- Access to the cluster as a user with the **cluster-admin** role.
- A cluster installed on bare metal infrastructure configured with the OpenShift SDN default CNI network provider.
- The cluster is in a known good state, without any errors.

Procedure

1. To backup the configuration for the cluster network, enter the following command:

```
$ oc get Network.config.openshift.io cluster -o yaml > cluster-openshift-sdn.yaml
```

2. To enable the migration, set an annotation on the Cluster Network Operator configuration object by entering the following command:

```
$ oc annotate Network.operator.openshift.io cluster \
  'networkoperator.openshift.io/network-migration'=""
```

3. To change the default CNI network provider, enter the following command:

```
$ oc patch Network.config.openshift.io cluster \
  --type='merge' --patch '{ "spec": { "networkType": "OVNKubernetes" } }'
```

4. To confirm the migration disabled the OpenShift SDN default CNI network provider and removed all OpenShift SDN pods, enter the following command. It might take several moments for all the OpenShift SDN pods to stop.

```
$ watch oc get pod -n openshift-sdn
```

5. To complete the migration, reboot each node in your cluster. For example, you could use a bash script similar to the following. The script assumes that you can connect to each host by using **ssh** and that you have configured **sudo** to not prompt for a password.

```
#!/bin/bash

for ip in $(oc get nodes  -o jsonpath='{.items[*].status.addresses[?
(@.type=="InternalIP")].address}')
do
   echo "reboot node $ip"
   ssh -o StrictHostKeyChecking=no core@$ip sudo shutdown -r -t 3
done
```

6. After the nodes in your cluster have rebooted, confirm that the migration succeeded:

   a. To confirm that the default CNI network provider is OVN-Kubernetes, enter the following command. The value of **status.networkType** must be **OVNKubernetes**.

   ```
   $ oc get network.config/cluster -o jsonpath='{.status.networkType}{"\n"}'
   ```

   b. To confirm that the cluster nodes are in the **Ready** state, enter the following command:

```
$ oc get nodes
```

If a node is stuck in the **NotReady** state, reboot the node again.

c. To confirm that your pods are not in an error state, enter the following command:

```
$ oc get pods --all-namespaces -o wide --sort-by='{.spec.nodeName}'
```

If pods on a node are in an error state, reboot that node.

7. Complete the following steps only if the migration succeeds and your cluster is in a good state:

a. To remove the migration annotation from the Cluster Network Operator configuration object, enter the following command:

```
$ oc annotate Network.operator.openshift.io cluster \
  networkoperator.openshift.io/network-migration-
```

b. To remove the OpenShift SDN network provider namespace, enter the following command:

```
$ oc delete namespace openshift-sdn
```

13.3. ROLLBACK TO THE OPENSHIFT SDN NETWORK PROVIDER

As a cluster administrator, you can rollback to the OpenShift SDN cluster default Container Network Interface (CNI) provider from the OVN-Kubernetes default CNI network provider if the migration to OVN-Kubernetes unsuccessful.

13.3.1. Rolling back the default CNI network provider to OpenShift SDN

As a cluster administrator, you can rollback your cluster to the OpenShift SDN default Container Network Interface (CNI) network provider. During the rollback, you must reboot every node in your cluster.

重要

Only rollback to OpenShift SDN if the migration to OVN-Kubernetes is unsuccessful.

Prerequisites

- Install the OpenShift CLI (**oc**).
- Access to the cluster as a user with the **cluster-admin** role.
- A cluster installed on bare metal infrastructure configured with the OVN-Kubernetes default CNI network provider.

1. To enable the migration, set an annotation on the Cluster Network Operator configuration object by entering the following command:

```
$ oc annotate Network.operator.openshift.io cluster \
  'networkoperator.openshift.io/network-migration'=""
```

2. To change the default CNI network provider, enter the following command:

   ```
   $ oc patch Network.config.openshift.io cluster \
     --type='merge' --patch '{ "spec": { "networkType": "OpenShiftSDN" } }'
   ```

3. Optional: Use the backup of the cluster network configuration that you created before the migration to restore any customizations to the network configuration that you might have made. To restore the customizations, enter the following command to edit the Cluster Network Operator configuration:

   ```
   $ oc edit Network.config.openshift.io cluster
   ```

4. To confirm that the migration disabled the OVN-Kubernetes default CNI network provider and removed all the OVN-Kubernetes pods, enter the following command. It might take several moments for all the OVN-Kubernetes pods to stop.

   ```
   $ watch oc get pod -n openshift-ovn-kubernetes
   ```

5. To complete the rollback, reboot each node in your cluster. For example, you could use a bash script similar to the following. The script assumes that you can connect to each host by using **ssh** and that you have configured **sudo** to not prompt for a password.

   ```
   #!/bin/bash

   for ip in $(oc get nodes  -o jsonpath='{.items[*].status.addresses[?
   (@.type=="InternalIP")].address}')
   do
      echo "reboot node $ip"
      ssh -o StrictHostKeyChecking=no core@$ip sudo shutdown -r -t 3
   done
   ```

6. After the nodes in your cluster have rebooted, enter the following command to confirm that the default CNI network provider is OpenShift SDN. The value of **status.networkType** must be **OpenShiftSDN**.

   ```
   $ oc get network.config/cluster -o jsonpath='{.status.networkType}{"\n"}'
   ```

7. To confirm that the OpenShift SDN pods are in the **READY** state, enter the following command:

   ```
   $ oc get pod -n openshift-sdn --watch
   ```

8. To remove the migration annotation from the Cluster Network Operator configuration object, enter the following command:

   ```
   $ oc annotate Network.operator.openshift.io cluster \
     networkoperator.openshift.io/network-migration-
   ```

9. To remove the OVN-Kubernetes network provider namespace, enter the following command:

   ```
   $ oc delete namespace openshift-ovn-kubernetes
   ```

13.4. ENABLING MULTICAST FOR A PROJECT

重要

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.

注意

In OpenShift Container Platform 4.5, a bug prevents Pods in the same namespace, but assigned to different nodes, from communicating over multicast. For more information, see BZ#1843695.

13.4.1. About multicast

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

重要

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

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

13.4.2. Enabling multicast between pods

You can enable multicast between pods for your project.

Prerequisites

- Install the OpenShift CLI (**oc**).
- You must log in to the cluster with a user that has the **cluster-admin** role.

Procedure

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

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

Verification

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

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

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

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

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

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

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

4. Start the multicast listener.

   a. Get the IP address for the Pod:

      ```
      $ POD_IP=$(oc get pods mlistener -o jsonpath='{.status.podIP}')
      ```

   b. To start the multicast listener, in a new terminal window or tab, enter the following command:

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

5. Start the multicast transmitter.

   a. Get the pod network IP address range:

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

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

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

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

   ```
   mlistener
   ```

13.5. DISABLING MULTICAST FOR A PROJECT

重要

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.

13.5.1. Disabling multicast between pods

You can disable multicast between pods for your project.

Prerequisites

- Install the OpenShift CLI (**oc**).
- You must log in to the cluster with a user that has the **cluster-admin** role.

Procedure

- Disable multicast by running the following command:

  ```
  $ oc annotate namespace <namespace> \ 1
      k8s.ovn.org/multicast-enabled-
  ```

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

第 14 章 CONFIGURING ROUTES

14.1. ROUTE CONFIGURATION

14.1.1. 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
```

14.1.2. 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**.

重要

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.

14.1.3. Troubleshooting throughput issues

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

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

- Use a packet analyzer, such as ping or tcpdump to analyze traffic between a pod and its node. For example, run the tcpdump tool on each pod while reproducing the behavior that led to the issue. Review the captures on both sides to compare send and receive timestamps to analyze the latency of traffic to and from a pod. Latency can occur in OpenShift Container Platform if a node interface is overloaded with traffic from other pods, storage devices, or the data plane.

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

  1. **podip** is the IP address for the pod. Run the **oc get pod <pod_name> -o wide** command to get the IP address of a pod.

  tcpdump generates a file at **/tmp/dump.pcap** containing all traffic between these two pods. Ideally, run the analyzer shortly before the issue is reproduced and stop the analyzer shortly after the issue is finished reproducing to minimize the size of the file. You can also run a packet analyzer between the nodes (eliminating the SDN from the equation) with:

  ```
  $ tcpdump -s 0 -i any -w /tmp/dump.pcap port 4789
  ```

- Use a bandwidth measuring tool, such as iperf, to measure streaming throughput and UDP throughput. Run the tool from the pods first, then from the nodes, to locate any bottlenecks.
  - For information on installing and using iperf, see this Red Hat Solution .

14.1.4. 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.

14.1.4.1. 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 specified cookie name:

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

where:

<route_name>

Specifies the name of the route.

<cookie_name>

Specifies the name for the cookie.

For example, to annotate the route **my_route** with the cookie name **my_cookie**:

```
$ oc annotate route my_route router.openshift.io/cookie_name="my_cookie"
```

2. Capture the route host name in a variable:

```
$ ROUTE_NAME=$(oc get route <route_name> -o jsonpath='{.spec.host}')
```

where:

<route_name>

Specifies the name of the route.

3. Save the cookie, and then access the route:

```
$ curl $ROUTE_NAME -k -c /tmp/cookie_jar
```

Use the cookie saved by the previous command when connecting to the route:

```
$ curl $ROUTE_NAME -k -b /tmp/cookie_jar
```

14.1.5. Path-based routes

Path-based routes specify a path component that can be compared against a URL, which requires that the traffic for the route be HTTP based. Thus, multiple routes can be served using the same host name, each with a different path. Routers should match routes based on the most specific path to the least. However, this depends on the router implementation.

The following table shows example routes and their accessibility:

表 14.1. Route availability

Route	When Compared to	Accessible
www.example.com/test	*www.example.com/test*	Yes
	www.example.com	No
www.example.com/test and *www.example.com*	*www.example.com/test*	Yes
	www.example.com	Yes
www.example.com	*www.example.com/text*	Yes (Matched by the host, not the route)
	www.example.com	Yes

An unsecured route with a path

```
apiVersion: v1
kind: Route
metadata:
  name: route-unsecured
spec:
  host: www.example.com
  path: "/test" 1
  to:
    kind: Service
    name: service-name
```

1 The path is the only added attribute for a path-based route.

注意

Path-based routing is not available when using passthrough TLS, as the router does not terminate TLS in that case and cannot read the contents of the request.

14.1.6. Route-specific annotations

The Ingress Controller can set the default options for all the routes it exposes. An individual route can override some of these defaults by providing specific configurations in its annotations. Red Hat does not support adding a route annotation to an operator-managed route.

重要

To create a whitelist with multiple source IPs or subnets, use a space-delimited list. Any other delimiter type causes the list to be ignored without a warning or error message.

表 14.2. 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**.
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.	

Variable	Description	Environment variable used as default
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**
haproxy.router.openshift.io/ip_whitelist	Sets a whitelist for the route. The whitelist is a space-separated list of IP addresses and CIDR ranges for the approved source addresses. Requests from IP addresses that are not in the whitelist are dropped. The maximum number of IP addresses and CIDR ranges allowed in a whitelist is 61.	
haproxy.router.openshift.io/hsts_header	Sets a Strict-Transport-Security header for the edge terminated or re-encrypt route.	
haproxy.router.openshift.io/log-send-hostname	Sets the **hostname** field in the Syslog header. Uses the host name of the system. **log-send-hostname** is enabled by default if any Ingress API logging method, such as sidecar or Syslog facility, is enabled for the router.	

注意

Environment variables cannot 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.

注意

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

A route that allows only one specific IP address

```
metadata:
  annotations:
    haproxy.router.openshift.io/ip_whitelist: 192.168.1.10
```

A route that allows several IP addresses

```
metadata:
  annotations:
    haproxy.router.openshift.io/ip_whitelist: 192.168.1.10 192.168.1.11 192.168.1.12
```

A route that allows an IP address CIDR network

```
metadata:
  annotations:
    haproxy.router.openshift.io/ip_whitelist: 192.168.1.0/24
```

A route that allows both IP an address and IP address CIDR networks

```
metadata:
  annotations:
    haproxy.router.openshift.io/ip_whitelist: 180.5.61.153 192.168.1.0/24 10.0.0.0/8
```

14.1.7. Configuring the route admission policy

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

警告

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

Prerequisites

- Cluster administrator privileges.

Procedure

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

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

Sample Ingress Controller configuration

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

14.2. SECURED ROUTES

Secure routes provide the ability to use several types of TLS termination to serve certificates to the client. The following sections describe how to create re-encrypt, edge, and passthrough routes with custom certificates.

重要

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.

14.2.1. Creating a re-encrypt route with a custom certificate

You can configure a secure route using reencrypt TLS termination with a custom certificate by using the **oc create route** command.

Prerequisites

- You must have a certificate/key pair in PEM-encoded files, where the certificate is valid for the route host.

- You may have a separate CA certificate in a PEM-encoded file that completes the certificate chain.

- You must have a separate destination CA certificate in a PEM-encoded file.

- You must have a service that you want to expose.

注意

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.

14.2.2. Creating an edge route with a custom certificate

You can configure a secure route using edge TLS termination with a custom certificate by using the **oc create route** command. With an edge route, the Ingress Controller terminates TLS encryption before forwarding traffic to the destination pod. The route specifies the TLS certificate and key that the Ingress Controller uses for the route.

Prerequisites

- You must have a certificate/key pair in PEM-encoded files, where the certificate is valid for the route host.

- You may have a separate CA certificate in a PEM-encoded file that completes the certificate chain.

- You must have a service that you want to expose.

注意

Password protected key files are not supported. To remove a passphrase from a key file, use the following command:

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

Procedure

This procedure creates a **Route** resource with a custom certificate and edge TLS termination. The following assumes that the certificate/key pair are in the **tls.crt** and **tls.key** files in the current working directory. You may also specify a CA certificate if needed to complete the certificate chain. Substitute the actual path names for **tls.crt**, **tls.key**, and (optionally) **ca.crt**. Substitute the name of the service that you want to expose for **frontend**. Substitute the appropriate 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.

14.2.3. Creating a passthrough route

You can configure a secure route using passthrough termination by using the **oc create route** command. With passthrough termination, encrypted traffic is sent straight to the destination without the router providing TLS termination. Therefore no key or certificate is required on the route.

Prerequisites

- You must have a service that you want to expose.

Procedure

- Create a **Route** resource:

```
$ oc create route passthrough route-passthrough-secured --service=frontend --port=8080
```

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

A Secured Route Using Passthrough Termination

```
apiVersion: v1
kind: Route
metadata:
  name: route-passthrough-secured 1
spec:
  host: www.example.com
  port:
    targetPort: 8080
  tls:
    termination: passthrough 2
    insecureEdgeTerminationPolicy: None 3
  to:
    kind: Service
    name: frontend
```

1 The name of the object, which is limited to 63 characters.

2 The **termination** field is set to **passthrough**. This is the only required **tls** field.

3 Optional **insecureEdgeTerminationPolicy**. The only valid values are **None**, **Redirect**, or empty for disabled.

The destination pod is responsible for serving certificates for the traffic at the endpoint. This is currently the only method that can support requiring client certificates, also known as two-way authentication.

第 15 章 CONFIGURING INGRESS CLUSTER TRAFFIC

15.1. CONFIGURING INGRESS CLUSTER TRAFFIC OVERVIEW

OpenShift Container Platform provides the following methods for communicating from outside the cluster with services running in the cluster.

The methods are recommended, in order or preference:

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

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

15.2. CONFIGURING EXTERNALIPS FOR SERVICES

As a cluster administrator, you can designate an IP address block that is external to the cluster that can send traffic to services in the cluster.

This functionality is generally most useful for clusters installed on bare-metal hardware.

15.2.1. Prerequisites

- Your network infrastructure must route traffic for the external IP addresses to your cluster.

15.2.2. About ExternalIP

For non-cloud environments, OpenShift Container Platform supports the assignment of external IP addresses to a **Service** object **spec.externalIPs[]** field through the **ExternalIP** facility. By setting this field, OpenShift Container Platform assigns an additional virtual IP address to the service. The IP address can be outside the service network defined for the cluster. A service configured with an ExternalIP functions similarly to a service with **type=NodePort**, allowing you to direct traffic to a local node for load balancing.

You must configure your networking infrastructure to ensure that the external IP address blocks that you define are routed to the cluster.

OpenShift Container Platform extends the ExternalIP functionality in Kubernetes by adding the following capabilities:

- Restrictions on the use of external IP addresses by users through a configurable policy
- Allocation of an external IP address automatically to a service upon request

警告

Disabled by default, use of ExternalIP functionality can be a security risk, because in-cluster traffic to an external IP address is directed to that service. This could allow cluster users to intercept sensitive traffic destined for external resources.

重要

This feature is supported only in non-cloud deployments. For cloud deployments, use the load balancer services for automatic deployment of a cloud load balancer to target the endpoints of a service.

You can assign an external IP address in the following ways:

Automatic assignment of an external IP

OpenShift Container Platform automatically assigns an IP address from the **autoAssignCIDRs** CIDR block to the **spec.externalIPs[]** array when you create a **Service** object with **spec.type=LoadBalancer** set. In this case, OpenShift Container Platform implements a non-cloud version of the load balancer service type and assigns IP addresses to the services. Automatic assignment is disabled by default and must be configured by a cluster administrator as described in the following section.

Manual assignment of an external IP

OpenShift Container Platform uses the IP addresses assigned to the **spec.externalIPs[]** array when you create a **Service** object. You cannot specify an IP address that is already in use by another service.

15.2.2.1. Configuration for ExternalIP

Use of an external IP address in OpenShift Container Platform is governed by the following fields in the **Network.config.openshift.io** CR named **cluster**:

- **spec.externalIP.autoAssignCIDRs** defines an IP address block used by the load balancer when choosing an external IP address for the service. OpenShift Container Platform supports only a single IP address block for automatic assignment. This can be simpler than having to manage the port space of a limited number of shared IP addresses when manually assigning ExternalIPs to services. If automatic assignment is enabled, a **Service** object with **spec.type=LoadBalancer** is allocated an external IP address.

- **spec.externalIP.policy** defines the permissible IP address blocks when manually specifying an IP address. OpenShift Container Platform does not apply policy rules to IP address blocks defined by **spec.externalIP.autoAssignCIDRs**.

If routed correctly, external traffic from the configured external IP address block can reach service endpoints through any TCP or UDP port that the service exposes.

重要

You must ensure that the IP address block you assign terminates at one or more nodes in your cluster.

OpenShift Container Platform supports both the automatic and manual assignment of IP addresses, and each address is guaranteed to be assigned to a maximum of one service. This ensures that each service can expose its chosen ports regardless of the ports exposed by other services.

注意

To use IP address blocks defined by **autoAssignCIDRs** in OpenShift Container Platform, you must configure the necessary IP address assignment and routing for your host network.

The following YAML describes a service with an external IP address configured:

Example Service object with spec.externalIPs[] set

```
apiVersion: v1
kind: Service
metadata:
  name: http-service
spec:
  clusterIP: 172.30.163.110
  externalIPs:
  - 192.168.132.253
  externalTrafficPolicy: Cluster
  ports:
  - name: highport
    nodePort: 31903
    port: 30102
    protocol: TCP
    targetPort: 30102
  selector:
    app: web
  sessionAffinity: None
  type: LoadBalancer
status:
  loadBalancer:
    ingress:
    - ip: 192.168.132.253
```

15.2.2.2. Restrictions on the assignment of an external IP address

As a cluster administrator, you can specify IP address blocks to allow and to reject.

Restrictions apply only to users without **cluster-admin** privileges. A cluster administrator can always set the service **spec.externalIPs[]** field to any IP address.

You configure IP address policy with a **policy** object defined by specifying the **spec.ExternalIP.policy** field. The policy object has the following shape:

```
{
  "policy": {
    "allowedCIDRs": [],
    "rejectedCIDRs": []
  }
}
```

When configuring policy restrictions, the following rules apply:

- If **policy={}** is set, then creating a **Service** object with **spec.ExternalIPs[]** set will fail. This is the default for OpenShift Container Platform.
- If **policy=null** is set, then creating a **Service** object with **spec.ExternalIPs[]** set to any IP address is allowed.
- If **policy** is set and either **policy.allowedCIDRs[]** or **policy.rejectedCIDRs[]** is set, the following rules apply:
  - If **allowedCIDRs[]** and **rejectedCIDRs[]** are both set, then **rejectedCIDRs[]** has precedence over **allowedCIDRs[]**.
  - If **allowedCIDRs[]** is set, creating a **Service** object with **spec.ExternalIPs[]** will succeed only if the specified IP addresses are allowed.
  - If **rejectedCIDRs[]** is set, creating a **Service** object with **spec.ExternalIPs[]** will succeed only if the specified IP addresses are not rejected.

15.2.2.3. Example policy objects

The examples that follow demonstrate several different policy configurations.

- In the following example, the policy prevents OpenShift Container Platform from creating any service with an external IP address specified:

  Example policy to reject any value specified for Service object spec.externalIPs[]

  ```
  apiVersion: config.openshift.io/v1
  kind: Network
  metadata:
    name: cluster
  spec:
    externalIP:
      policy: {}
    ...
  ```

- In the following example, both the **allowedCIDRs** and **rejectedCIDRs** fields are set.

  Example policy that includes both allowed and rejected CIDR blocks

```
apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  externalIP:
    policy:
      allowedCIDRs:
      - 172.16.66.10/23
      rejectedCIDRs:
      - 172.16.66.10/24
  ...
```

- In the following example, **policy** is set to **null**. If set to **null**, when inspecting the configuration object by entering **oc get networks.config.openshift.io -o yaml**, the **policy** field will not appear in the output.

Example policy to allow any value specified for **Service object spec.externalIPs[]**

```
apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  externalIP:
    policy: null
  ...
```

15.2.3. ExternalIP address block configuration

The configuration for ExternalIP address blocks is defined by a Network custom resource (CR) named **cluster**. The Network CR is part of the **config.openshift.io** API group.

重要

During cluster installation, the Cluster Version Operator (CVO) automatically creates a Network CR named **cluster**. Creating any other CR objects of this type is not supported.

The following YAML describes the ExternalIP configuration:

Network.config.openshift.io CR named **cluster**

```
apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  externalIP:
    autoAssignCIDRs: [] 1
    policy: 2
      ...
```

1 Defines the IP address block in CIDR format that is available for automatic assignment of external

IP addresses to a service. Only a single IP address range is allowed.

2 Defines restrictions on manual assignment of an IP address to a service. If no restrictions are defined, specifying the **spec.externalIP** field in a **Service** object is not allowed. By default, no restrictions are defined.

The following YAML describes the fields for the **policy** stanza:

Network.config.openshift.io policy stanza

```
policy:
  allowedCIDRs: [] 1
  rejectedCIDRs: [] 2
```

1 A list of allowed IP address ranges in CIDR format.

2 A list of rejected IP address ranges in CIDR format.

Example external IP configurations

Several possible configurations for external IP address pools are displayed in the following examples:

- The following YAML describes a configuration that enables automatically assigned external IP addresses:

  #### Example configuration with **spec.externalIP.autoAssignCIDRs** set

  ```
  apiVersion: config.openshift.io/v1
  kind: Network
  metadata:
    name: cluster
  spec:
    ...
    externalIP:
      autoAssignCIDRs:
      - 192.168.132.254/29
  ```

- The following YAML configures policy rules for the allowed and rejected CIDR ranges:

  #### Example configuration with **spec.externalIP.policy** set

  ```
  apiVersion: config.openshift.io/v1
  kind: Network
  metadata:
    name: cluster
  spec:
    ...
    externalIP:
      policy:
        allowedCIDRs:
        - 192.168.132.0/29
        - 192.168.132.8/29
        rejectedCIDRs:
        - 192.168.132.7/32
  ```

15.2.4. Configure external IP address blocks for your cluster

As a cluster administrator, you can configure the following ExternalIP settings:

- An ExternalIP address block used by OpenShift Container Platform to automatically populate the **spec.clusterIP** field for a **Service** object.
- A policy object to restrict what IP addresses may be manually assigned to the **spec.clusterIP** array of a **Service** object.

Prerequisites

- Install the OpenShift CLI (**oc**).
- Access to the cluster as a user with the **cluster-admin** role.

Procedure

1. Optional: To display the current external IP configuration, enter the following command:

```
$ oc describe networks.config cluster
```

2. To edit the configuration, enter the following command:

```
$ oc edit networks.config cluster
```

3. Modify the ExternalIP configuration, as in the following example:

```
apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  ...
  externalIP: 1
  ...
```

1 Specify the configuration for the **externalIP** stanza.

4. To confirm the updated ExternalIP configuration, enter the following command:

```
$ oc get networks.config cluster -o go-template='{{.spec.externalIP}}{{"\n"}}'
```

15.2.5. Next steps

- Configuring ingress cluster traffic for a service external IP

15.3. CONFIGURING INGRESS CLUSTER TRAFFIC USING AN INGRESS CONTROLLER

OpenShift Container Platform provides methods for communicating from outside the cluster with services running in the cluster. This method uses an Ingress Controller.

15.3.1. Using Ingress Controllers and routes

The Ingress Operator manages Ingress Controllers and wildcard DNS.

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

An Ingress Controller is configured to accept external requests and proxy them based on the configured routes. This is limited to HTTP, HTTPS using SNI, and TLS using SNI, which is sufficient for web applications and services that work over TLS with SNI.

Work with your administrator to configure an Ingress Controller to accept external requests and proxy them based on the configured routes.

The administrator can create a wildcard DNS entry and then set up an Ingress Controller. Then, you can work with the edge Ingress Controller without having to contact the administrators.

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.

注意

The procedures in this section require prerequisites performed by the cluster administrator.

15.3.2. Prerequisites

Before starting the following procedures, the administrator must:

- Set up the external port to the cluster networking environment so that requests can reach the cluster.
- Make sure there is at least one user with cluster admin role. To add this role to a user, run the following command:

```
$ oc adm policy add-cluster-role-to-user cluster-admin username
```

- Have an OpenShift Container Platform cluster with at least one master and at least one node and a system outside the cluster that has network access to the cluster. This procedure assumes that the external system is on the same subnet as the cluster. The additional networking required for external systems on a different subnet is out-of-scope for this topic.

15.3.3. Creating a project and service

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

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

Prerequisites

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

Procedure

1. Create a new project for your service:

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

   For example:

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

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

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

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

   ```
   $ oc get svc -n myproject
   ```

   #### Example output

   ```
   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.

15.3.4. Exposing the service by creating a route

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

Procedure

To expose the service:

1. Log in to OpenShift Container Platform.

2. Log in to the project where the service you want to expose is located:

   ```
   $ oc project myproject
   ```

3. Run the following command to expose the route:

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

   For example:

```
$ oc expose service mysql-80-rhel7
```

Example output

```
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 uses 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
```

Example output

```
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.

MySQL [(none)]>
```

15.3.5. Configuring Ingress Controller sharding by using route labels

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

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**.

15.3.6. Configuring Ingress Controller sharding by using namespace labels

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

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

Procedure

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

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

Example output

```
apiVersion: 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**.

15.3.7. 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.

15.4. CONFIGURING INGRESS CLUSTER TRAFFIC USING A LOAD BALANCER

OpenShift Container Platform provides methods for communicating from outside the cluster with services running in the cluster. This method uses a load balancer.

15.4.1. Using a load balancer to get traffic into the cluster

If you do not need a specific external IP address, you can configure a load balancer service to allow external access to an OpenShift Container Platform cluster.

A load balancer service allocates a unique IP. The load balancer has a single edge router IP, which can be a virtual IP (VIP), but is still a single machine for initial load balancing.

注意

If a pool is configured, it is done at the infrastructure level, not by a cluster administrator.

注意

The procedures in this section require prerequisites performed by the cluster administrator.

15.4.2. Prerequisites

Before starting the following procedures, the administrator must:

- Set up the external port to the cluster networking environment so that requests can reach the cluster.
- Make sure there is at least one user with cluster admin role. To add this role to a user, run the following command:

  ```
  $ oc adm policy add-cluster-role-to-user cluster-admin username
  ```

- Have an OpenShift Container Platform cluster with at least one master and at least one node and a system outside the cluster that has network access to the cluster. This procedure assumes that the external system is on the same subnet as the cluster. The additional networking required for external systems on a different subnet is out-of-scope for this topic.

15.4.3. Creating a project and service

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

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

Prerequisites

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

Procedure

1. Create a new project for your service:

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

   For example:

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

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

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

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

   ```
   $ oc get svc -n myproject
   ```

   Example output

   ```
   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.

15.4.4. Exposing the service by creating a route

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

Procedure

To expose the service:

1. Log in to OpenShift Container Platform.

2. Log in to the project where the service you want to expose is located:

   ```
   $ oc project myproject
   ```

3. Run the following command to expose the route:

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

   For example:

   ```
   $ oc expose service mysql-80-rhel7
   ```

   Example output

   ```
   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 uses 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
   ```

   Example output

   ```
   Enter password:
   Welcome to the MariaDB monitor.  Commands end with ; or \g.

   MySQL [(none)]>
   ```

15.4.5. Creating a load balancer service

Use the following procedure to create a load balancer service.

Prerequisites

- Make sure that the project and service you want to expose exist.

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
```

Example output

```
NAME       TYPE           CLUSTER-IP       EXTERNAL-IP                              PORT(S)
AGE
egress-2   LoadBalancer   172.30.22.226   ad42f5d8b303045-487804948.example.com
3306:30357/TCP   15m
```

The service has an external IP address automatically assigned if there is a cloud provider enabled.

7. On the master, use a tool, such as cURL, to make sure you can reach the service using the public IP address:

```
$ curl <public-ip>:<port>
```

For example:

```
$ curl 172.29.121.74:3306
```

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

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

```
$ mysql -h 172.30.131.89 -u admin -p
```

Example output

```
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.

MySQL [(none)]>
```

15.5. CONFIGURING INGRESS CLUSTER TRAFFIC FOR A SERVICE EXTERNAL IP

You can attach an external IP address to a service so that it is available to traffic outside the cluster. This is generally useful only for a cluster installed on bare metal hardware. The external network infrastructure must be configured correctly to route traffic to the service.

15.5.1. Prerequisites

- Your cluster is configured with ExternalIPs enabled. For more information, read Configuring ExternalIPs for services.

15.5.2. Attaching an ExternalIP to a service

You can attach an ExternalIP to a service. If your cluster is configured to allocate an ExternalIP automatically, you might not need to manually attach an ExternalIP to the service.

Procedure

1. Optional: To confirm what IP address ranges are configured for use with ExternalIP, enter the following command:

```
$ oc get networks.config cluster -o jsonpath='{.spec.externalIP}{"\n"}'
```

If **autoAssignCIDRs** is set, OpenShift Container Platform automatically assigns an ExternalIP to a new **Service** object if the **spec.externalIPs** field is not specified.

2. Attach an ExternalIP to the service.

   a. If you are creating a new service, specify the **spec.externalIPs** field and provide an array of one or more valid IP addresses. For example:

```
apiVersion: v1
kind: Service
metadata:
  name: svc-with-externalip
spec:
  ...
  externalIPs:
  - 192.174.120.10
```

   b. If you are attaching an ExternalIP to an existing service, enter the following command. Replace **<name>** with the service name. Replace **<ip_address>** with a valid ExternalIP address. You can provide multiple IP addresses separated by commas.

```
$ oc patch svc <name> -p \
  '{
    "spec": {
      "externalIPs": [ "<ip_address>" ]
    }
  }'
```

   For example:

```
$ oc patch svc mysql-55-rhel7 -p '{"spec":{"externalIPs":["192.174.120.10"]}}'
```

   Example output

```
"mysql-55-rhel7" patched
```

3. To confirm that an ExternalIP address is attached to the service, enter the following command. If you specified an ExternalIP for a new service, you must create the service first.

```
$ oc get svc
```

Example output

```
NAME             CLUSTER-IP      EXTERNAL-IP     PORT(S)    AGE
mysql-55-rhel7   172.30.131.89   192.174.120.10  3306/TCP   13m
```

15.5.3. Additional resources

- Configuring ExternalIPs for services

15.6. 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**.

15.6.1. 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.

> **重要**
>
> Using a node port requires additional port resources.

A **NodePort** exposes the service on a static port on the node's IP address. **NodePort**s are in the **30000** to **32767** range by default, which means a **NodePort** is unlikely to match a service's intended port. For example, port **8080** may be exposed as port **31020** on the node.

The administrator must ensure the external IP addresses are routed to the nodes.

NodePorts and external IPs are independent and both can be used concurrently.

> **注意**
>
> The procedures in this section require prerequisites performed by the cluster administrator.

15.6.2. Prerequisites

Before starting the following procedures, the administrator must:

- Set up the external port to the cluster networking environment so that requests can reach the cluster.
- Make sure there is at least one user with cluster admin role. To add this role to a user, run the following command:

  ```
  $ oc adm policy add-cluster-role-to-user cluster-admin <user_name>
  ```

- Have an OpenShift Container Platform cluster with at least one master and at least one node and a system outside the cluster that has network access to the cluster. This procedure assumes that the external system is on the same subnet as the cluster. The additional networking required for external systems on a different subnet is out-of-scope for this topic.

15.6.3. Creating a project and service

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

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

Prerequisites

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

Procedure

1. Create a new project for your service:

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

For example:

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

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

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

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

```
$ oc get svc -n myproject
```

Example output

```
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.

15.6.4. Exposing the service by creating a route

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

Procedure

To expose the service:

1. Log in to OpenShift Container Platform.

2. Log in to the project where the service you want to expose is located:

```
$ oc project myproject
```

3. To expose a node port for the application, 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
```

Example output

```
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
```

第 16 章 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.

重要

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 virual network, for a supported provider.

16.1. PREREQUISITES

- Review the sites that your cluster requires access to and determine whether any of them must bypass the proxy. By default, all cluster system egress traffic is proxied, including calls to the cloud provider API for the cloud that hosts your cluster. System-wide proxy affects system components only, not user workloads. Add sites to the Proxy object's **spec.noProxy** field to bypass the proxy if necessary.

  注意

  The Proxy object **status.noProxy** field is populated with the values of the **networking.machineNetwork[].cidr**, **networking.clusterNetwork[].cidr**, and **networking.serviceNetwork[]** fields from your installation configuration.

  For installations on Amazon Web Services (AWS), Google Cloud Platform (GCP), Microsoft Azure, and Red Hat OpenStack Platform (RHOSP), the **Proxy** object **status.noProxy** field is also populated with the instance metadata endpoint (**169.254.169.254**).

16.2. 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.

注意

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.

   注意

   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.

③ A comma-separated list of destination domain names, domains, IP addresses or other network CIDRs to exclude proxying.

Preface a domain with **.** to match subdomains only. For example, **.y.com** matches **x.y.com**, but not **y.com**. Use ***** to bypass proxy for all destinations. If you scale up workers that are not included in the network defined by the **networking.machineNetwork[].cidr** field from the installation configuration, you must add them to this list to prevent connection issues.

This field is ignored if neither the **httpProxy** or **httpsProxy** fields are set.

④ One or more URLs external to the cluster to use to perform a readiness check before writing the **httpProxy** and **httpsProxy** values to status.

⑤ A reference to the 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.

注意

The URL scheme must be **http**. The **https** scheme is currently not supported.

16.3. REMOVING THE CLUSTER-WIDE PROXY

The **cluster** Proxy object cannot be deleted. To remove the proxy from a cluster, remove all **spec** fields from the Proxy object.

Prerequisites

- Cluster administrator permissions
- OpenShift Container Platform **oc** CLI tool installed

Procedure

1. Use the **oc edit** command to modify the proxy:

```
$ oc edit proxy/cluster
```

2. Remove all **spec** fields from the Proxy object. For example:

```
apiVersion: config.openshift.io/v1
kind: Proxy
metadata:
  name: cluster
spec: {}
status: {}
```

3. Save the file to apply the changes.

第 17 章 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.

注意

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.

17.1. 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.

注意

The **Proxy** object **status.noProxy** field is populated with the values of the **networking.machineNetwork[].cidr**, **networking.clusterNetwork[].cidr**, and **networking.serviceNetwork[]** fields from your installation configuration.

For installations on Amazon Web Services (AWS), Google Cloud Platform (GCP), Microsoft Azure, and Red Hat OpenStack Platform (RHOSP), the **Proxy** object **status.noProxy** field is also populated with the instance metadata endpoint (**169.254.169.254**).

Procedure

1. Edit your **install-config.yaml** file and add the proxy settings. For example:

```
apiVersion: v1
baseDomain: my.domain.com
proxy:
  httpProxy: http://<username>:<pswd>@<ip>:<port> 1
  httpsProxy: 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. 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 match subdomains only. For example, **.y.com** matches **x.y.com**, but not **y.com**. Use * to bypass proxy for all destinations.

4 If provided, the installation program generates a config map that is named **user-ca-bundle** in the **openshift-config** namespace that contains one or more additional CA certificates that are required for proxying HTTPS connections. The Cluster Network Operator then creates a **trusted-ca-bundle** config map that merges these contents with the Red Hat Enterprise Linux CoreOS (RHCOS) trust bundle, and this config map is referenced in the **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.

注意

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**.

注意

Only the **Proxy** object named **cluster** is supported, and no additional proxies can be created.

17.2. 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.

注意

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.

注意

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.

3 A comma-separated list of destination domain names, domains, IP addresses or other network CIDRs to exclude proxying.

Preface a domain with **.** to match subdomains only. For example, **.y.com** matches **x.y.com**, but not **y.com**. Use * to bypass proxy for all destinations. If you scale up workers that are not included in the network defined by the **networking.machineNetwork[].cidr** field from the installation configuration, you must add them to this list to prevent connection issues.

This field is ignored if neither the **httpProxy** or **httpsProxy** fields are set.

4 One or more URLs external to the cluster to use to perform a readiness check before writing the **httpProxy** and **httpsProxy** values to status.

5 A reference to the 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.

注意

The URL scheme must be **http**. The **https** scheme is currently not supported.

17.3. 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.

注意

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.

第 18 章 LOAD BALANCING ON RHOSP

18.1. USING THE OCTAVIA OVN LOAD BALANCER PROVIDER DRIVER WITH KURYR SDN

If your OpenShift Container Platform cluster uses Kuryr and was installed on a Red Hat OpenStack Platform (RHOSP) 13 cloud that was later upgraded to RHOSP 16, you can configure it to use the Octavia OVN provider driver.

重要

Kuryr replaces existing load balancers after you change provider drivers. This process results in some downtime.

Prerequisites

- Install the RHOSP CLI, **openstack**.
- Install the OpenShift Container Platform CLI, **oc**.
- Verify that the Octavia OVN driver on RHOSP is enabled.

提示

To view a list of available Octavia drivers, on a command line, enter **openstack loadbalancer provider list**.

The **ovn** driver is displayed in the command's output.

Procedure

To change from the Octavia Amphora provider driver to Octavia OVN:

1. Open the **kuryr-config** ConfigMap. On a command line, enter:

```
$ oc -n openshift-kuryr edit cm kuryr-config
```

2. In the ConfigMap, delete the line that contains **kuryr-octavia-provider: default**. For example:

```
...
kind: ConfigMap
metadata:
  annotations:
    networkoperator.openshift.io/kuryr-octavia-provider: default 1
...
```

1 Delete this line. The cluster will regenerate it with **ovn** as the value.

Wait for the Cluster Network Operator to detect the modification and to redeploy the **kuryr-controller** and **kuryr-cni** pods. This process might take several minutes.

3. Verify that the **kuryr-config** ConfigMap annotation is present with **ovn** as its value. On a command line, enter:

   ```
   $ oc -n openshift-kuryr edit cm kuryr-config
   ```

   The **ovn** provider value is displayed in the output:

   ```
   ...
   kind: ConfigMap
   metadata:
     annotations:
       networkoperator.openshift.io/kuryr-octavia-provider: ovn
   ...
   ```

4. Verify that RHOSP recreated its load balancers.

   a. On a command line, enter:

      ```
      $ openstack loadbalancer list | grep amphora
      ```

      A single Amphora load balancer is displayed. For example:

      ```
      a4db683b-2b7b-4988-a582-c39daaad7981 | ostest-7mbj6-kuryr-api-loadbalancer  |
      84c99c906edd475ba19478a9a6690efd | 172.30.0.1     | ACTIVE              | amphora
      ```

   b. Search for **ovn** load balancers by entering:

      ```
      $ openstack loadbalancer list | grep ovn
      ```

      The remaining load balancers of the **ovn** type are displayed. For example:

      ```
      2dffe783-98ae-4048-98d0-32aa684664cc | openshift-apiserver-operator/metrics |
      84c99c906edd475ba19478a9a6690efd | 172.30.167.119 | ACTIVE              | ovn
      0b1b2193-251f-4243-af39-2f99b29d18c5 | openshift-etcd/etcd                  |
      84c99c906edd475ba19478a9a6690efd | 172.30.143.226 | ACTIVE              | ovn
      f05b07fc-01b7-4673-bd4d-adaa4391458e | openshift-dns-operator/metrics       |
      84c99c906edd475ba19478a9a6690efd | 172.30.152.27  | ACTIVE              | ovn
      ```

18.2. SCALING CLUSTERS FOR APPLICATION TRAFFIC BY USING OCTAVIA

OpenShift Container Platform clusters that run on Red Hat OpenStack Platform (RHOSP) can use the Octavia load balancing service to distribute traffic across multiple virtual machines (VMs) or floating IP addresses. This feature mitigates the bottleneck that single machines or addresses create.

If your cluster uses Kuryr, the Cluster Network Operator created an internal Octavia load balancer at deployment. You can use this load balancer for application network scaling.

If your cluster does not use Kuryr, you must create your own Octavia load balancer to use it for application network scaling.

18.2.1. Scaling clusters by using Octavia

If you want to use multiple API load balancers, or if your cluster does not use Kuryr, create an Octavia load balancer and then configure your cluster to use it.

Prerequisites

- Octavia is available on your Red Hat OpenStack Platform (RHOSP) deployment.

Procedure

1. From a command line, create an Octavia load balancer that uses the Amphora driver:

```
$ openstack loadbalancer create --name API_OCP_CLUSTER --vip-subnet-id
<id_of_worker_vms_subnet>
```

   You can use a name of your choice instead of **API_OCP_CLUSTER**.

2. After the load balancer becomes active, create listeners:

```
$ openstack loadbalancer listener create --name API_OCP_CLUSTER_6443 --protocol
HTTPS--protocol-port 6443 API_OCP_CLUSTER
```

   注意

   To view the status of the load balancer, enter **openstack loadbalancer list**.

3. Create a pool that uses the round robin algorithm and has session persistence enabled:

```
$ openstack loadbalancer pool create --name API_OCP_CLUSTER_pool_6443 --lb-
algorithm ROUND_ROBIN --session-persistence type=<source_IP_address> --listener
API_OCP_CLUSTER_6443 --protocol HTTPS
```

4. To ensure that control plane machines are available, create a health monitor:

```
$ openstack loadbalancer healthmonitor create --delay 5 --max-retries 4 --timeout 10 --type
TCP API_OCP_CLUSTER_pool_6443
```

5. Add the control plane machines as members of the load balancer pool:

```
$ for SERVER in $(MASTER-0-IP MASTER-1-IP MASTER-2-IP)
do
  openstack loadbalancer member create --address $SERVER  --protocol-port 6443
API_OCP_CLUSTER_pool_6443
done
```

6. Optional: To reuse the cluster API floating IP address, unset it:

```
$ openstack floating ip unset $API_FIP
```

7. Add either the unset **API_FIP** or a new address to the created load balancer VIP:

```
$ openstack floating ip set  --port $(openstack loadbalancer show -c <vip_port_id> -f value
API_OCP_CLUSTER) $API_FIP
```

Your cluster now uses Octavia for load balancing.

注意

If Kuryr uses the Octavia Amphora driver, all traffic is routed through a single Amphora virtual machine (VM).

You can repeat this procedure to create additional load balancers, which can alleviate the bottleneck.

18.2.2. Scaling clusters that use Kuryr by using Octavia

If your cluster uses Kuryr, associate the API floating IP address of your cluster with the pre-existing Octavia load balancer.

Prerequisites

- Your OpenShift Container Platform cluster uses Kuryr.
- Octavia is available on your Red Hat OpenStack Platform (RHOSP) deployment.

Procedure

1. Optional: From a command line, to reuse the cluster API floating IP address, unset it:

```
$ openstack floating ip unset $API_FIP
```

2. Add either the unset **API_FIP** or a new address to the created load balancer VIP:

```
$ openstack floating ip set --port $(openstack loadbalancer show -c <vip_port_id> -f value
${OCP_CLUSTER}-kuryr-api-loadbalancer) $API_FIP
```

Your cluster now uses Octavia for load balancing.

注意

If Kuryr uses the Octavia Amphora driver, all traffic is routed through a single Amphora virtual machine (VM).

You can repeat this procedure to create additional load balancers, which can alleviate the bottleneck.

18.3. SCALING FOR INGRESS TRAFFIC BY USING RHOSP OCTAVIA

You can use Octavia load balancers to scale Ingress controllers on clusters that use Kuryr.

Prerequisites

- Your OpenShift Container Platform cluster uses Kuryr.
- Octavia is available on your RHOSP deployment.

Procedure

1. To copy the current internal router service, on a command line, enter:

   ```
   $ oc -n openshift-ingress get svc router-internal-default -o yaml > external_router.yaml
   ```

2. In the file **external_router.yaml**, change the values of **metadata.name** and **spec.type** to **LoadBalancer**.

   ### Example router file

   ```
   apiVersion: v1
   kind: Service
   metadata:
     labels:
       ingresscontroller.operator.openshift.io/owning-ingresscontroller: default
     name: router-external-default 1
     namespace: openshift-ingress
   spec:
     ports:
     - name: http
       port: 80
       protocol: TCP
       targetPort: http
     - name: https
       port: 443
       protocol: TCP
       targetPort: https
     - name: metrics
       port: 1936
       protocol: TCP
       targetPort: 1936
     selector:
       ingresscontroller.operator.openshift.io/deployment-ingresscontroller: default
     sessionAffinity: None
     type: LoadBalancer 2
   ```

   1 Ensure that this value is descriptive, like **router-external-default**.

   2 Ensure that this value is **LoadBalancer**.

注意

You can delete timestamps and other information that is irrelevant to load balancing.

1. From a command line, create a service from the **external_router.yaml** file:

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

2. Verify that the external IP address of the service is the same as the one that is associated with the load balancer:

   a. On a command line, retrieve the external IP address of the service:

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

Example output

```
NAME                    TYPE           CLUSTER-IP       EXTERNAL-IP    PORT(S)
AGE
router-external-default   LoadBalancer   172.30.235.33    10.46.22.161
80:30112/TCP,443:32359/TCP,1936:30317/TCP   3m38s
router-internal-default   ClusterIP      172.30.115.123   <none>
80/TCP,443/TCP,1936/TCP                      22h
```

b. Retrieve the IP address of the load balancer:

```
$ openstack loadbalancer list | grep router-external
```

Example output

```
| 21bf6afe-b498-4a16-a958-3229e83c002c | openshift-ingress/router-external-default |
66f3816acf1b431691b8d132cc9d793c | 172.30.235.33  | ACTIVE | octavia |
```

c. Verify that the addresses you retrieved in the previous steps are associated with each other in the floating IP list:

```
$ openstack floating ip list | grep 172.30.235.33
```

Example output

```
| e2f80e97-8266-4b69-8636-e58bacf1879e | 10.46.22.161 | 172.30.235.33 | 655e7122-
806a-4e0a-a104-220c6e17bda6 | a565e55a-99e7-4d15-b4df-f9d7ee8c9deb |
66f3816acf1b431691b8d132cc9d793c |
```

You can now use the value of **EXTERNAL-IP** as the new Ingress address.

注意

If Kuryr uses the Octavia Amphora driver, all traffic is routed through a single Amphora virtual machine (VM).

You can repeat this procedure to create additional load balancers, which can alleviate the bottleneck.