OpenShift Container Platform 4.18

Networking Operators

Managing networking-specific Operators in OpenShift Container Platform

Last Updated: 2026-04-01

OpenShift Container Platform 4.18 Networking Operators

Managing networking-specific Operators in OpenShift Container Platform

Legal Notice

Abstract

This document covers the installation, configuration, and management of various networking-related Operators in OpenShift Container Platform.

Table of Contents

CHAPTER 1. KUBERNETES NMSTATE OPERATOR **7**
1.1. INSTALLING THE KUBERNETES NMSTATE OPERATOR 8
1.1.1. Installing the Kubernetes NMState Operator by using the web console 8
1.1.2. Installing the Kubernetes NMState Operator by using the CLI 9
1.1.3. Viewing metrics collected by the Kubernetes NMState Operator 10
1.2. UNINSTALLING THE KUBERNETES NMSTATE OPERATOR 12
1.3. ADDITIONAL RESOURCES 14
1.4. NEXT STEPS 15

CHAPTER 2. AWS LOAD BALANCER OPERATOR **16**
2.1. AWS LOAD BALANCER OPERATOR RELEASE NOTES 16
2.1.1. AWS Load Balancer Operator 1.2.0 16
2.1.2. AWS Load Balancer Operator 1.1.1 16
2.1.3. AWS Load Balancer Operator 1.1.0 16
2.1.4. AWS Load Balancer Operator 1.0.1 17
2.1.5. AWS Load Balancer Operator 1.0.0 17
2.1.6. Earlier versions 18
2.2. AWS LOAD BALANCER OPERATOR IN OPENSHIFT CONTAINER PLATFORM 18
2.2.1. AWS Load Balancer Operator considerations 18
2.2.2. Deploying the AWS Load Balancer Operator 19
2.2.3. Using the AWS Load Balancer Operator in an AWS VPC cluster extended into an Outpost 19
2.3. PREPARING AN AWS STS CLUSTER FOR THE AWS LOAD BALANCER OPERATOR 20
2.3.1. The IAM role for the AWS Load Balancer Operator 21
2.3.1.1. Creating an AWS IAM role by using the Cloud Credential Operator utility 21
2.3.1.2. Creating an AWS IAM role by using the AWS CLI 22
2.3.2. Configuring the ARN role for the AWS Load Balancer Operator 24
2.3.3. The IAM role for the AWS Load Balancer Controller 25
2.3.3.1. Creating an AWS IAM role for the controller by using the Cloud Credential Operator utility 25
2.3.3.2. Creating an AWS IAM role for the controller by using the AWS CLI 26
2.3.4. Additional resources 28
2.4. INSTALLING THE AWS LOAD BALANCER OPERATOR 29
2.4.1. Installing the AWS Load Balancer Operator by using the web console 29
2.4.2. Installing the AWS Load Balancer Operator by using the CLI 30
2.4.3. Creating the AWS Load Balancer Controller 31
2.5. CONFIGURING THE AWS LOAD BALANCER OPERATOR 35
2.5.1. Trusting the certificate authority of the cluster-wide proxy 35
2.5.2. Adding TLS termination on the AWS Load Balancer 36
2.5.3. Creating multiple ingress resources through a single AWS Load Balancer 37
2.5.4. AWS Load Balancer Operator logs 41

CHAPTER 3. EBPF MANAGER OPERATOR **42**
3.1. ABOUT THE EBPF MANAGER OPERATOR 42
3.1.1. About Extended Berkeley Packet Filter (eBPF) 42
3.1.2. About the eBPF Manager Operator 42
3.1.3. Additional resources 43
3.2. INSTALLING THE EBPF MANAGER OPERATOR 43
3.2.1. Installing the eBPF Manager Operator using the CLI 43
3.2.2. Installing the eBPF Manager Operator using the web console 45
3.3. DEPLOYING AN EBPF PROGRAM 46
3.3.1. Deploying a containerized eBPF program 46

CHAPTER 4. EXTERNAL DNS OPERATOR **49**

4.1. EXTERNAL DNS OPERATOR RELEASE NOTES 49
4.1.1. External DNS Operator 1.3 49
4.1.2. External DNS Operator 1.2 49
4.1.3. External DNS Operator 1.1 50
4.1.4. External DNS Operator 1.0 50
4.2. UNDERSTANDING THE EXTERNAL DNS OPERATOR 51
4.2.1. External DNS Operator domain name limitations 51
4.2.2. Deploying the External DNS Operator 52
4.2.3. Viewing External DNS Operator logs 52
4.3. INSTALLING THE EXTERNAL DNS OPERATOR 52
4.3.1. Installing the External DNS Operator with OperatorHub 52
4.3.2. Installing the External DNS Operator by using the CLI 53
4.4. EXTERNAL DNS OPERATOR CONFIGURATION PARAMETERS 55
4.4.1. External DNS Operator configuration parameters 55
4.5. CREATING DNS RECORDS ON AWS 58
4.5.1. Creating DNS records on a public hosted zone for AWS by using Red Hat External DNS Operator 58
4.5.2. Creating DNS records in a different AWS account by using a shared VPC 60
4.6. CREATING DNS RECORDS ON AZURE 61
4.6.1. Creating DNS records on an Azure DNS zone 61
4.7. CREATING DNS RECORDS ON GOOGLE CLOUD PLATFORM 63
4.7.1. Creating DNS records on a public managed zone for Google Cloud 64
4.8. CREATING DNS RECORDS ON INFOBLOX 66
4.8.1. Creating DNS records on a public DNS zone on Infoblox 66
4.9. CONFIGURING THE CLUSTER-WIDE PROXY ON THE EXTERNAL DNS OPERATOR 67
4.9.1. Trusting the certificate authority of the cluster-wide proxy 67

CHAPTER 5. METALLB OPERATOR 69
5.1. ABOUT METALLB AND THE METALLB OPERATOR 69
5.1.1. When to use MetalLB 69
5.1.2. MetalLB Operator custom resources 69
5.1.3. MetalLB software components 70
5.1.4. MetalLB and external traffic policy 71
5.1.5. MetalLB concepts for layer 2 mode 72
5.1.6. MetalLB concepts for BGP mode 74
5.1.7. Limitations and restrictions 76
5.1.7.1. Infrastructure considerations for MetalLB 76
5.1.7.2. Limitations for layer 2 mode 76
5.1.7.2.1. Single-node bottleneck 76
5.1.7.2.2. Slow failover performance 77
5.1.7.2.3. Additional Network and MetalLB cannot use same network 77
5.1.7.3. Limitations for BGP mode 77
5.1.7.3.1. Node failure can break all active connections 77
5.1.7.3.2. Support for a single ASN and a single router ID only 78
5.1.8. Additional resources 78
5.2. INSTALLING THE METALLB OPERATOR 78
5.2.1. Installing the MetalLB Operator from the OperatorHub by using the web console 78
5.2.2. Installing from OperatorHub using the CLI 79
5.2.3. Starting MetalLB on your cluster 81
5.2.4. Deployment specifications for MetalLB 82
5.2.4.1. Limit speaker pods to specific nodes 82
5.2.4.2. Configuring pod priority and pod affinity in a MetalLB deployment 83
5.2.4.3. Configuring pod CPU limits in a MetalLB deployment 85
5.2.5. Additional resources 86

5.3. UPGRADING THE METALLB OPERATOR 86
5.3.1. Manually upgrading the MetalLB Operator 86
5.3.2. Additional resources 87

CHAPTER 6. CLUSTER NETWORK OPERATOR IN OPENSHIFT CONTAINER PLATFORM 88
6.1. CLUSTER NETWORK OPERATOR 88
6.2. VIEWING THE CLUSTER NETWORK CONFIGURATION 88
6.3. VIEWING CLUSTER NETWORK OPERATOR STATUS 89
6.4. ENABLING IP FORWARDING GLOBALLY 90
6.5. VIEWING CLUSTER NETWORK OPERATOR LOGS 91
6.6. CLUSTER NETWORK OPERATOR CONFIGURATION 91
6.6.1. Cluster Network Operator configuration object 91
6.6.2. defaultNetwork object configuration 92
6.6.3. Configuration for the OVN-Kubernetes network plugin 93
6.6.4. Cluster Network Operator example configuration 98
6.7. ADDITIONAL RESOURCES 98

CHAPTER 7. DNS OPERATOR IN OPENSHIFT CONTAINER PLATFORM 99
7.1. CHECKING THE STATUS OF THE DNS OPERATOR 99
7.2. VIEW THE DEFAULT DNS 99
7.3. USING DNS FORWARDING 100
7.4. CHECKING DNS OPERATOR STATUS 102
7.5. VIEWING DNS OPERATOR LOGS 103
7.6. SETTING THE COREDNS LOG LEVEL 103
7.7. VIEWING THE COREDNS LOGS 104
7.8. SETTING THE COREDNS OPERATOR LOG LEVEL 104
7.9. TUNING THE COREDNS CACHE 105
7.9.1. Changing the DNS Operator managementState 106
7.9.2. Controlling DNS pod placement 107
7.9.3. Configuring DNS forwarding with TLS 109

CHAPTER 8. INGRESS OPERATOR IN OPENSHIFT CONTAINER PLATFORM 112
8.1. OPENSHIFT CONTAINER PLATFORM INGRESS OPERATOR 112
8.2. THE INGRESS CONFIGURATION ASSET 112
8.3. INGRESS CONTROLLER CONFIGURATION PARAMETERS 112
8.3.1. Ingress Controller TLS security profiles 123
8.3.1.1. Understanding TLS security profiles 123
8.3.1.2. Configuring the TLS security profile for the Ingress Controller 125
8.3.1.3. Configuring mutual TLS authentication 127
8.4. VIEW THE DEFAULT INGRESS CONTROLLER 128
8.5. VIEW INGRESS OPERATOR STATUS 128
8.6. VIEW INGRESS CONTROLLER LOGS 128
8.7. VIEW INGRESS CONTROLLER STATUS 129
8.8. CREATING A CUSTOM INGRESS CONTROLLER 129
8.9. CONFIGURING THE INGRESS CONTROLLER 130
8.9.1. Setting a custom default certificate 130
8.9.2. Removing a custom default certificate 131
8.9.3. Autoscaling an Ingress Controller 132
8.9.4. Scaling an Ingress Controller 136
8.9.5. Configuring Ingress access logging 137
8.9.6. Setting Ingress Controller thread count 141
8.9.7. Configuring an Ingress Controller to use an internal load balancer 141
8.9.8. Configuring global access for an Ingress Controller on Google Cloud 143
8.9.9. Setting the Ingress Controller health check interval 144

8.9.10. Configuring the default Ingress Controller for your cluster to be internal 145
8.9.11. Configuring the route admission policy 145
8.9.12. Using wildcard routes 146
8.9.13. HTTP header configuration 147
8.9.14. Setting or deleting HTTP request and response headers in an Ingress Controller 150
8.9.15. Using X-Forwarded headers 151
8.9.15.1. Example use cases 152
8.9.16. Enable or disable HTTP/2 on Ingress Controllers 152
8.9.16.1. Enabling HTTP/2 153
8.9.16.2. Disabling HTTP/2 154
8.9.17. Configuring the PROXY protocol for an Ingress Controller 155
8.9.18. Specifying an alternative cluster domain using the appsDomain option 157
8.9.19. Converting HTTP header case 158
8.9.20. Using router compression 160
8.9.21. Exposing router metrics 161
8.9.22. Customizing HAProxy error code response pages 162
8.9.23. Setting the Ingress Controller maximum connections 165
8.10. ADDITIONAL RESOURCES 165

CHAPTER 9. INGRESS NODE FIREWALL OPERATOR IN OPENSHIFT CONTAINER PLATFORM 166
9.1. INGRESS NODE FIREWALL OPERATOR 166
9.2. INSTALLING THE INGRESS NODE FIREWALL OPERATOR 166
9.3. INSTALLING THE INGRESS NODE FIREWALL OPERATOR USING THE WEB CONSOLE 167
9.4. DEPLOYING INGRESS NODE FIREWALL OPERATOR 169
9.5. INGRESS NODE FIREWALL CONFIGURATION OBJECT 169
9.5.1. Ingress Node Firewall Operator example configuration 170
9.5.2. Ingress Node Firewall rules object 171
9.5.2.1. Ingress object configuration 171
9.5.2.2. Ingress Node Firewall rules object example 172
9.5.2.3. Zero trust Ingress Node Firewall rules object example 173
9.6. INGRESS NODE FIREWALL OPERATOR INTEGRATION 174
9.7. CONFIGURING INGRESS NODE FIREWALL OPERATOR TO USE THE EBPF MANAGER OPERATOR 174
9.8. VIEWING INGRESS NODE FIREWALL OPERATOR RULES 175
9.9. TROUBLESHOOTING THE INGRESS NODE FIREWALL OPERATOR 175
9.10. ADDITIONAL RESOURCES 176

CHAPTER 10. SR-IOV OPERATOR 177
10.1. INSTALLING THE SR-IOV NETWORK OPERATOR 177
10.1.1. Using the CLI to install the SR-IOV Network Operator 177
10.1.2. Using the web console to install the SR-IOV Network Operator 178
10.1.3. Additional resources 179
10.2. CONFIGURING THE SR-IOV NETWORK OPERATOR 179
10.2.1. Configuring the SR-IOV Network Operator 180
10.2.2. SR-IOV Network Operator config custom resource 180
10.2.3. About the Network Resources Injector 182
10.2.4. Disabling or enabling the Network Resources Injector 183
10.2.5. About the SR-IOV Network Operator admission controller webhook 184
10.2.6. Disabling or enabling the SR-IOV Network Operator admission controller webhook 185
10.2.7. Configuring a custom NodeSelector for the SR-IOV Network Config daemon 185
10.2.8. Configuring the SR-IOV Network Operator for single node installations 186
10.2.8.1. Deploying the SR-IOV Operator for hosted control planes 187
10.2.9. About the SR-IOV network metrics exporter 188
10.2.9.1. Enabling the SR-IOV network metrics exporter 190

10.3. UNINSTALLING THE SR-IOV NETWORK OPERATOR 191
10.3.1. Uninstalling the SR-IOV Network Operator 191
10.3.2. Additional resources 192

CHAPTER 1. KUBERNETES NMSTATE OPERATOR

The Kubernetes NMState Operator provides a Kubernetes API for performing state-driven network configuration across the OpenShift Container Platform cluster's nodes with NMState. The Kubernetes NMState Operator provides users with functionality to configure various network interface types, DNS, and routing on cluster nodes. Additionally, the daemons on the cluster nodes periodically report on the state of each node's network interfaces to the API server.

IMPORTANT

Red Hat supports the Kubernetes NMState Operator in production environments on bare-metal, IBM Power®, IBM Z®, IBM® LinuxONE, VMware vSphere, and Red Hat OpenStack Platform (RHOSP) installations.

Red Hat support exists for using the Kubernetes NMState Operator on Microsoft Azure but in a limited capacity. Support is limited to configuring DNS servers on your system as a postinstallation task.

Before you can use NMState with OpenShift Container Platform, you must install the Kubernetes NMState Operator. After you install the Kubernetes NMState Operator, you can complete the following tasks:

- Observing and updating the node network state and configuration
- Creating a manifest object that includes a customized **br-ex** bridge For more information on these tasks, see the *Additional resources* section

Before you can use NMState with OpenShift Container Platform, you must install the Kubernetes NMState Operator.

NOTE

The Kubernetes NMState Operator updates the network configuration of a secondary NIC. The Operator cannot update the network configuration of the primary NIC, or update the **br-ex** bridge on most on-premise networks.

On a bare-metal platform, using the Kubernetes NMState Operator to update the **br-ex** bridge network configuration is only supported if you set the **br-ex** bridge as the interface in a machine config manifest file. To update the **br-ex** bridge as a postinstallation task, you must set the **br-ex** bridge as the interface in the NMState configuration of the **NodeNetworkConfigurationPolicy** custom resource (CR) for your cluster. For more information, see Creating a manifest object that includes a customized br-ex bridge in *Postinstallation configuration*.

OpenShift Container Platform uses **nmstate** to report on and configure the state of the node network. This makes it possible to modify the network policy configuration, such as by creating a Linux bridge on all nodes, by applying a single configuration manifest to the cluster.

Node networking is monitored and updated by the following objects:

NodeNetworkState

Reports the state of the network on that node.

NodeNetworkConfigurationPolicy

Describes the requested network configuration on nodes. You update the node network configuration, including adding and removing interfaces, by applying a **NodeNetworkConfigurationPolicy** CR to the cluster.

NodeNetworkConfigurationEnactment

Reports the network policies enacted upon each node.

> NOTE
>
> Do not make configuration changes to the **br-ex** bridge or its underlying interfaces as a postinstallation task.

1.1. INSTALLING THE KUBERNETES NMSTATE OPERATOR

You can install the Kubernetes NMState Operator by using the web console or the CLI.

1.1.1. Installing the Kubernetes NMState Operator by using the web console

You can install the Kubernetes NMState Operator by using the web console. After you install the Kubernetes NMState Operator, the Operator has deployed the NMState State Controller as a daemon set across all of the cluster nodes.

Prerequisites

- You are logged in as a user with **cluster-admin** privileges.

Procedure

1. Select **Operators → OperatorHub**.
2. In the search field below **All Items**, enter **nmstate** and click **Enter** to search for the Kubernetes NMState Operator.
3. Click on the Kubernetes NMState Operator search result.
4. Click on **Install** to open the **Install Operator** window.
5. Click **Install** to install the Operator.
6. After the Operator finishes installing, click **View Operator**.
7. Under **Provided APIs**, click **Create Instance** to open the dialog box for creating an instance of **kubernetes-nmstate**.
8. In the **Name** field of the dialog box, ensure the name of the instance is **nmstate.**

   > NOTE
   >
   > The name restriction is a known issue. The instance is a singleton for the entire cluster.

9. Accept the default settings and click **Create** to create the instance.

1.1.2. Installing the Kubernetes NMState Operator by using the CLI

You can install the Kubernetes NMState Operator by using the OpenShift CLI (**oc)**. After it is installed, the Operator deploys the NMState State Controller as a daemon set across all of the cluster nodes to manage the node network state and configuration.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You are logged in as a user with **cluster-admin** privileges.

Procedure

1. Create the **nmstate** Operator namespace:

```
$ cat << EOF | oc apply -f -
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-nmstate
spec:
  finalizers:
  - kubernetes
EOF
```

2. Create the **OperatorGroup**:

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

3. Subscribe to the **nmstate** Operator:

```
$ cat << EOF| oc apply -f -
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: kubernetes-nmstate-operator
  namespace: openshift-nmstate
spec:
  channel: stable
  installPlanApproval: Automatic
  name: kubernetes-nmstate-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
EOF
```

4. Confirm the **ClusterServiceVersion** (CSV) status for the **nmstate** Operator deployment equals **Succeeded**:

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

5. Create an instance of the **nmstate** Operator:

```
$ cat << EOF | oc apply -f -
apiVersion: nmstate.io/v1
kind: NMState
metadata:
  name: nmstate
EOF
```

6. If your cluster has problems with the DNS health check probe because of DNS connectivity issues, you can add the following DNS host name configuration to the **NMState** CRD to build in health checks that can resolve these issues:

```
apiVersion: nmstate.io/v1
kind: NMState
metadata:
  name: nmstate
spec:
  probeConfiguration:
    dns:
      host: redhat.com
# ...
```

   a. Apply the DNS host name configuration to your cluster network by running the following command. Ensure that you replace **<filename>** with the name of your CRD file.

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

   b. Monitor the **nmstate** CRD until the resource reaches the **Available** condition by running the following command. Ensure that you set a value for the **--timeout** option so that if the **Available** condition is not met within this set maximum waiting time, the command times out and generates an error message.

   ```
   $ oc wait --for=condition=Available nmstate/nmstate --timeout=600s
   ```

Verification

1. Verify that all pods for the NMState Operator have the **Running** status by entering the following command:

```
$ oc get pod -n openshift-nmstate
```

1.1.3. Viewing metrics collected by the Kubernetes NMState Operator

The Kubernetes NMState Operator, **kubernetes-nmstate-operator**, can collect metrics from the **kubernetes_nmstate_features_applied** component and expose them as ready-to-use metrics. As a use case for viewing metrics, consider a situation where you created a **NodeNetworkConfigurationPolicy** custom resource (CR) and you want to confirm that the policy is active.

NOTE

The **kubernetes_nmstate_features_applied** metrics are not an API and might change between OpenShift Container Platform versions.

In the **Developer** and **Administrator** perspectives, the Metrics UI includes some predefined CPU, memory, bandwidth, and network packet queries for the selected project. You can run custom Prometheus Query Language (PromQL) queries for CPU, memory, bandwidth, network packet and application metrics for the project.

The following example demonstrates a **NodeNetworkConfigurationPolicy** manifest example that is applied to an OpenShift Container Platform cluster:

```
# ...
interfaces:
  - name: br1
    type: linux-bridge
    state: up
    ipv4:
      enabled: true
      dhcp: true
      dhcp-custom-hostname: foo
    bridge:
      options:
        stp:
          enabled: false
      port: []
# ...
```

The **NodeNetworkConfigurationPolicy** manifest exposes metrics and makes them available to the Cluster Monitoring Operator (CMO). The following example shows some exposed metrics:

```
controller_runtime_reconcile_time_seconds_bucket{controller="nodenetworkconfigurationenactment",le
="0.005"} 16
controller_runtime_reconcile_time_seconds_bucket{controller="nodenetworkconfigurationenactment",le
="0.01"} 16
controller_runtime_reconcile_time_seconds_bucket{controller="nodenetworkconfigurationenactment",le
="0.025"} 16
...
# HELP kubernetes_nmstate_features_applied Number of nmstate features applied labeled by its
name
# TYPE kubernetes_nmstate_features_applied gauge
kubernetes_nmstate_features_applied{name="dhcpv4-custom-hostname"} 1
```

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have logged in to the web console as the administrator and installed the Kubernetes NMState Operator.

- You have access to the cluster as a developer or as a user with view permissions for the project that you are viewing metrics for.
- You have enabled monitoring for user-defined projects.
- You have deployed a service in a user-defined project.
- You have created a **NodeNetworkConfigurationPolicy** manifest and applied it to your cluster.

Procedure

1. If you want to view the metrics from the **Developer** perspective in the OpenShift Container Platform web console, complete the following tasks:
    a. Click **Observe**.
    b. To view the metrics of a specific project, select the project in the **Project:** list. For example, **openshift-nmstate**.
    c. Click the **Metrics** tab.
    d. To visualize the metrics on the plot, select a query from the **Select query** list or create a custom PromQL query based on the selected query by selecting **Show PromQL**.

    > **NOTE**
    >
    > In the **Developer** perspective, you can only run one query at a time.

2. If you want to view the metrics from the **Administrator** perspective in the OpenShift Container Platform web console, complete the following tasks:
    a. Click **Observe → Metrics**.
    b. Enter **kubernetes_nmstate_features_applied** in the **Expression** field.
    c. Click **Add query** and then **Run queries**.
3. To explore the visualized metrics, do any of the following tasks:
    a. To zoom into the plot and change the time range, do any of the following tasks:
        - To visually select the time range, click and drag on the plot horizontally.
        - To select the time range, use the menu which is in the upper left of the console.
    b. To reset the time range, select **Reset zoom**.
    c. To display the output for all the queries at a specific point in time, hold the mouse cursor on the plot at that point. The query output displays in a pop-up box.

1.2. UNINSTALLING THE KUBERNETES NMSTATE OPERATOR

Remove the Kubernetes NMState Operator and related resources when they are no longer needed.

You can use the Operator Lifecycle Manager (OLM) to uninstall the Kubernetes NMState Operator, but by design OLM does not delete any associated custom resource definitions (CRDs), custom resources (CRs), or API Services.

Before you uninstall the Kubernetes NMState Operator from the **Subcription** resource used by OLM, identify what Kubernetes NMState Operator resources to delete. This identification ensures that you can delete resources without impacting your running cluster.

If you need to reinstall the Kubernetes NMState Operator, see "Installing the Kubernetes NMState Operator by using the CLI" or "Installing the Kubernetes NMState Operator by using the web console".

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have installed the **jq** CLI tool.
- You are logged in as a user with **cluster-admin** privileges.

Procedure

1. Unsubscribe the Kubernetes NMState Operator from the **Subcription** resource by running the following command:

```
$ oc delete --namespace openshift-nmstate subscription kubernetes-nmstate-operator
```

2. Find the **ClusterServiceVersion** (CSV) resource that associates with the Kubernetes NMState Operator:

```
$ oc get --namespace openshift-nmstate clusterserviceversion
```

Example output that lists a CSV resource

```
NAME                              DISPLAY                       VERSION   REPLACES     PHASE
kubernetes-nmstate-operator.v4.18.0   Kubernetes NMState Operator   4.18.0
Succeeded
```

3. Delete the CSV resource. After you delete the file, OLM deletes certain resources, such as **RBAC**, that it created for the Operator.

```
$ oc delete --namespace openshift-nmstate clusterserviceversion kubernetes-nmstate-operator.v4.18.0
```

4. Delete the **nmstate** CR and any associated **Deployment** resources by running the following commands:

```
$ oc -n openshift-nmstate delete nmstate nmstate
```

```
$ oc delete --all deployments --namespace=openshift-nmstate
```

5. After you deleted the **nmstate** CR, remove the **nmstate-console-plugin** console plugin name from the **console.operator.openshift.io/cluster** CR.

a. Store the position of the **nmstate-console-plugin** entry that exists among the list of enable plugins by running the following command. The following command uses the **jq** CLI tool to store the index of the entry in an environment variable named **INDEX**:

```
INDEX=$(oc get console.operator.openshift.io cluster -o json | jq -r '.spec.plugins |
to_entries[] | select(.value == "nmstate-console-plugin") | .key')
```

b. Remove the **nmstate-console-plugin** entry from the **console.operator.openshift.io/cluster** CR by running the following patch command:

```
$ oc patch console.operator.openshift.io cluster --type=json -p "[{\"op\": \"remove\",
\"path\": \"/spec/plugins/$INDEX\"}]"
```

- **INDEX** is an auxiliary variable. You can specify a different name for this variable.

6. Optional: To preserve CR instances so that you can restore them after you delete CRDs, enter the following command:

```
$ oc get -A nncp -o yaml > cluster-nncp.yaml
```

IMPORTANT

To reuse preserved CRs, such as NNCPs, you must uninstall the Kubernetes NMState Operator, reinstall the Kubernetes NMState Operator, and then run the following command to restore the CRs:

```
$ oc apply -f cluster-nncp.yaml
```

7. Delete all the CRDs, such as **nmstates.nmstate.io**, by running the following commands:

```
$ oc delete crd nmstates.nmstate.io
```

```
$ oc delete crd nodenetworkconfigurationenactments.nmstate.io
```

```
$ oc delete crd nodenetworkstates.nmstate.io
```

```
$ oc delete crd nodenetworkconfigurationpolicies.nmstate.io
```

8. Delete the namespace:

```
$ oc delete namespace openshift-nmstate
```

1.3. ADDITIONAL RESOURCES

- Creating an interface on nodes
- Creating a manifest object that includes a customized br-ex bridge (Installer-provisioned infrastructure)

- Creating a manifest object that includes a customized br-ex bridge (User-provisioned infrastructure)

1.4. NEXT STEPS

- Observing and updating the node network state and configuration

CHAPTER 2. AWS LOAD BALANCER OPERATOR

2.1. AWS LOAD BALANCER OPERATOR RELEASE NOTES

The release notes for the AWS Load Balancer (ALB) Operator summarize all new features and enhancements, notable technical changes, major corrections from the previous version, and any known bugs upon general availability.

IMPORTANT

The AWS Load Balancer (ALB) Operator is only supported on the **x86_64** architecture.

These release notes track the development of the AWS Load Balancer Operator in OpenShift Container Platform.

For an overview of the AWS Load Balancer Operator, see AWS Load Balancer Operator in OpenShift Container Platform.

NOTE

AWS Load Balancer Operator currently does not support AWS GovCloud.

2.1.1. AWS Load Balancer Operator 1.2.0

The AWS Load Balancer Operator 1.2.0 release notes summarize all new features and enhancements, notable technical changes, major corrections from the previous version, and any known bugs upon general availability.

The following advisory is available for the AWS Load Balancer Operator version 1.2.0:

- RHEA-2025:0034 Release of AWS Load Balancer Operator 1.2.z on OperatorHub

Notable changes

- This release supports the AWS Load Balancer Controller version 2.8.2.
- With this release, the platform tags defined in the **Infrastructure** resource are added to all AWS objects created by the controller.

2.1.2. AWS Load Balancer Operator 1.1.1

The AWS Load Balancer Operator 1.1.1 release notes summarize all new features and enhancements, notable technical changes, major corrections from the previous version, and any known bugs upon general availability.

The following advisory is available for the AWS Load Balancer Operator version 1.1.1:

- RHEA-2024:0555 Release of AWS Load Balancer Operator 1.1.z on OperatorHub

2.1.3. AWS Load Balancer Operator 1.1.0

The AWS Load Balancer Operator 1.1.0 release notes summarize all new features and enhancements, notable technical changes, major corrections from the previous version, and any known bugs upon general availability.

The AWS Load Balancer Operator version 1.1.0 supports the AWS Load Balancer Controller version 2.4.4.

The following advisory is available for the AWS Load Balancer Operator version 1.1.0:

- RHEA-2023:6218 Release of AWS Load Balancer Operator on OperatorHub Enhancement Advisory Update

  Notable changes

- This release uses the Kubernetes API version 0.27.2.

  New features

- The AWS Load Balancer Operator now supports a standardized Security Token Service (STS) flow by using the Cloud Credential Operator.

  Bug fixes

- A FIPS-compliant cluster must use TLS version 1.2. Previously, webhooks for the AWS Load Balancer Controller only accepted TLS 1.3 as the minimum version, resulting in an error such as the following on a FIPS-compliant cluster:

  ```
  remote error: tls: protocol version not supported
  ```

  Now, the AWS Load Balancer Controller accepts TLS 1.2 as the minimum TLS version, resolving this issue. (OCPBUGS-14846)

2.1.4. AWS Load Balancer Operator 1.0.1

The AWS Load Balancer Operator 1.0.1 release notes summarize all new features and enhancements, notable technical changes, major corrections from the previous version, and any known bugs upon general availability.

The following advisory is available for the AWS Load Balancer Operator version 1.0.1:

- Release of AWS Load Balancer Operator 1.0.1 on OperatorHub

2.1.5. AWS Load Balancer Operator 1.0.0

The AWS Load Balancer Operator 1.0.0 release notes summarize all new features and enhancements, notable technical changes, major corrections from the previous version, and any known bugs upon general availability.

The AWS Load Balancer Operator is now generally available with this release. The AWS Load Balancer Operator version 1.0.0 supports the AWS Load Balancer Controller version 2.4.4.

The following advisory is available for the AWS Load Balancer Operator version 1.0.0:

- RHEA-2023:1954 Release of AWS Load Balancer Operator on OperatorHub Enhancement Advisory Update

IMPORTANT

The AWS Load Balancer (ALB) Operator version 1.x.x cannot upgrade automatically from the Technology Preview version 0.x.x. To upgrade from an earlier version, you must uninstall the ALB operands and delete the **aws-load-balancer-operator** namespace.

Notable changes

- This release uses the new **v1** API version.

Bug fixes

- Previously, the controller provisioned by the AWS Load Balancer Operator did not properly use the configuration for the cluster-wide proxy. These settings are now applied appropriately to the controller. (OCPBUGS-4052, OCPBUGS-5295)

2.1.6. Earlier versions

To evaluate the AWS Load Balancer Operator, use the two earliest versions, which are available as a Technology Preview. Do not use these versions in a production cluster.

For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

The following advisory is available for the AWS Load Balancer Operator version 0.2.0:

- RHEA-2022:9084 Release of AWS Load Balancer Operator on OperatorHub Enhancement Advisory Update

The following advisory is available for the AWS Load Balancer Operator version 0.0.1:

- RHEA-2022:5780 Release of AWS Load Balancer Operator on OperatorHub Enhancement Advisory Update

2.2. AWS LOAD BALANCER OPERATOR IN OPENSHIFT CONTAINER PLATFORM

To deploy and manage the AWS Load Balancer Controller, install the AWS Load Balancer Operator from the software catalog by using the OpenShift Container Platform web console or CLI. You can use the Operator to integrate AWS load balancers directly into your cluster infrastructure.

2.2.1. AWS Load Balancer Operator considerations

To ensure a successful deployment, review the limitations of the AWS Load Balancer Operator. Understanding these constraints helps avoid compatibility issues and ensures the Operator meets your architectural requirements before installation.

Review the following limitations before installing and using the AWS Load Balancer Operator:

- The IP traffic mode only works on AWS Elastic Kubernetes Service (EKS). The AWS Load Balancer Operator disables the IP traffic mode for the AWS Load Balancer Controller. As a result of disabling the IP traffic mode, the AWS Load Balancer Controller cannot use the pod readiness gate.

- The AWS Load Balancer Operator adds command-line flags such as **--disable-ingress-class-annotation** and **--disable-ingress-group-name-annotation** to the AWS Load Balancer Controller. Therefore, the AWS Load Balancer Operator does not allow using the **kubernetes.io/ingress.class** and **alb.ingress.kubernetes.io/group.name** annotations in the **Ingress** resource.
- The AWS Load Balancer Operator requires that the service type is **NodePort** and not **LoadBalancer** or **ClusterIP**.

2.2.2. Deploying the AWS Load Balancer Operator

After you deploy the The AWS Load Balancer Operator, the Operator automatically tags public subnets if the **kubernetes.io/role/elb** tag is missing. The Operator then identifies specific network resources in the underlying AWS cloud to ensure successful cluster integration.

The AWS Load Balancer Operator detects the following information from the underlying AWS cloud:

- The ID of the virtual private cloud (VPC) on which the cluster hosting the Operator is deployed.
- Public and private subnets of the discovered VPC.

The AWS Load Balancer Operator supports the Kubernetes service resource of type **LoadBalancer** by using Network Load Balancer (NLB) with the **instance** target type only.

Procedure

1. To deploy the AWS Load Balancer Operator on-demand from OperatorHub, create a **Subscription** object by running the following command:

```
$ oc -n aws-load-balancer-operator get sub aws-load-balancer-operator --
template='{{.status.installplan.name}}{{"\n"}}'
```

2. Check if the status of an install plan is **Complete** by running the following command:

```
$ oc -n aws-load-balancer-operator get ip <install_plan_name> --template='{{.status.phase}}
{{"\n"}}'
```

3. View the status of the **aws-load-balancer-operator-controller-manager** deployment by running the following command:

```
$ oc get -n aws-load-balancer-operator deployment/aws-load-balancer-operator-controller-
manager
```

Example output

```
NAME                                           READY   UP-TO-DATE   AVAILABLE   AGE
aws-load-balancer-operator-controller-manager  1/1     1            1           23h
```

2.2.3. Using the AWS Load Balancer Operator in an AWS VPC cluster extended into an Outpost

To provision an AWS Application Load Balancer in an AWS VPC cluster extended into an Outpost, configure the AWS Load Balancer Operator. Note that the Operator cannot provision AWS Network Load Balancers because AWS Outposts does not support them.

You can create an AWS Application Load Balancer either in the cloud subnet or in the Outpost subnet.

An Application Load Balancer in the cloud can attach to cloud-based compute nodes. An Application Load Balancer in the Outpost can attach to edge compute nodes.

You must annotate Ingress resources with the Outpost subnet or the VPC subnet, but not both.

Prerequisites

- You have extended an AWS VPC cluster into an Outpost.
- You have installed the OpenShift CLI (**oc**).
- You have installed the AWS Load Balancer Operator and created the AWS Load Balancer Controller.

Procedure

- Configure the **Ingress** resource to use a specified subnet:

  Example Ingress resource configuration

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: <application_name>
  annotations:
    alb.ingress.kubernetes.io/subnets: <subnet_id>
spec:
  ingressClassName: alb
  rules:
    - http:
        paths:
          - path: /
            pathType: Exact
            backend:
              service:
                name: <application_name>
                port:
                  number: 80
```

  where:

  <subnet_id>
  : Specifies the subnet to use. To use the Application Load Balancer in an Outpost, specify the Outpost subnet ID. To use the Application Load Balancer in the cloud, you must specify at least two subnets in different availability zones.

2.3. PREPARING AN AWS STS CLUSTER FOR THE AWS LOAD BALANCER OPERATOR

To install the Amazon Web Services (AWS) Load Balancer Operator on a cluster that uses the Security Token Service (STS), prepare the cluster by configuring the **CredentialsRequest** object. This ensures the Operator can bootstrap the AWS Load Balancer Controller and access the required secrets.

The AWS Load Balancer Operator waits until the required secrets are created and available.

Before you start any Security Token Service (STS) procedures, ensure that you meet the following prerequisites:

- You installed the OpenShift CLI (**oc**).
- You know the infrastructure ID of your cluster. To show this ID, run the following command in your CLI:

  ```
  $ oc get infrastructure cluster -o=jsonpath="{.status.infrastructureName}"
  ```

- You know the OpenID Connect (OIDC) DNS information for your cluster. To show this information, enter the following command in your CLI:

  ```
  $ oc get authentication.config cluster -o=jsonpath="{.spec.serviceAccountIssuer}"
  ```

  where:

  {.spec.serviceAccountIssuer}
  Specifies an OIDC DNS URL. An example URL is **https://rh-oidc.s3.us-east-1.amazonaws.com/28292va7ad7mr9r4he1fb09b14t59t4f**.

- You logged into the AWS management console, navigated to **IAM → Access management → Identity providers**, and located the OIDC Amazon Resource Name (ARN) information. An OIDC ARN example is **arn:aws:iam::777777777777:oidc-provider/<oidc_dns_url>**.

Additional resources

- the Cloud Credential Operator utility (**ccoctl**)

2.3.1. The IAM role for the AWS Load Balancer Operator

To install the Amazon Web Services (AWS) Load Balancer Operator on a cluster by using STS, configure an additional Identity and Access Management (IAM) role. This role enables the Operator to interact with subnets and Virtual Private Clouds (VPCs), allowing the Operator to generate the **CredentialsRequest** object required for bootstrapping.

You can create the IAM role by using the following options:

- Using the Cloud Credential Operator utility (**ccoctl**) and a predefined **CredentialsRequest** object.
- Using the AWS CLI and predefined AWS manifests.

Use the AWS CLI if your environment does not support the **ccoctl** command.

2.3.1.1. Creating an AWS IAM role by using the Cloud Credential Operator utility

To enable the AWS Load Balancer Operator to interact with subnets and VPCs, create an AWS IAM role by using the Cloud Credential Operator utility (**ccoctl**). By doing this task, you can generate the necessary credentials for the operator to function correctly within the cluster environment.

Prerequisites

- You must extract and prepare the **ccoctl** binary.

Procedure

1. Download the **CredentialsRequest** custom resource (CR) and store it in a directory by running the following command:

```
$ curl --create-dirs -o <credentials_requests_dir>/operator.yaml
https://raw.githubusercontent.com/openshift/aws-load-balancer-operator/main/hack/operator-credentials-request.yaml
```

2. Use the **ccoctl** utility to create an AWS IAM role by running the following command:

```
$ ccoctl aws create-iam-roles \
    --name <name> \
    --region=<aws_region> \
    --credentials-requests-dir=<credentials_requests_dir> \
    --identity-provider-arn <oidc_arn>
```

Example output

```
2023/09/12 11:38:57 Role arn:aws:iam::777777777777:role/<name>-aws-load-balancer-operator-aws-load-balancer-operator created
2023/09/12 11:38:57 Saved credentials configuration to: /home/user/<credentials_requests_dir>/manifests/aws-load-balancer-operator-aws-load-balancer-operator-credentials.yaml
2023/09/12 11:38:58 Updated Role policy for Role <name>-aws-load-balancer-operator-aws-load-balancer-operator created
```

where:

<name>

Specifies the Amazon Resource Name (ARN) for an AWS IAM role that was created for the AWS Load Balancer Operator, such as **arn:aws:iam::777777777777:role/<name>-aws-load-balancer-operator-aws-load-balancer-operator**.

> **NOTE**
>
> The length of an AWS IAM role name must be less than or equal to 12 characters.

2.3.1.2. Creating an AWS IAM role by using the AWS CLI

To enable the AWS Load Balancer Operator to interact with subnets and VPCs, create an AWS IAM role by using the AWS CLI. This enables the Operator to access and manage the necessary network resources within the cluster.

Prerequisites

- You must have access to the AWS Command Line Interface (**aws**).

Procedure

1. Generate a trust policy file by using your identity provider by running the following command:

```
$ cat <<EOF > albo-operator-trust-policy.json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Federated": "<oidc_arn>"
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringEquals": {
                    "<cluster_oidc_endpoint>:sub": "system:serviceaccount:aws-load-balancer-operator:aws-load-balancer-operator-controller-manager"
                }
            }
        }
    ]
}
EOF
```

where:

<oidc_arn>

Specifies the Amazon Resource Name (ARN) of the OIDC identity provider, such as **arn:aws:iam::777777777777:oidc-provider/rh-oidc.s3.us-east-1.amazonaws.com/28292va7ad7mr9r4he1fb09b14t59t4f**.

serviceaccount

Specifies the service account for the AWS Load Balancer Controller. An example of **<cluster_oidc_endpoint>** is **rh-oidc.s3.us-east-1.amazonaws.com/28292va7ad7mr9r4he1fb09b14t59t4f**.

2. Create the IAM role with the generated trust policy by running the following command:

```
$ aws iam create-role --role-name albo-operator --assume-role-policy-document file://albo-operator-trust-policy.json
```

Example output

```
ROLE arn:aws:iam::<aws_account_number>:role/albo-operator 2023-08-02T12:13:22Z (1)
ASSUMEROLEPOLICYDOCUMENT 2012-10-17
```

```
STATEMENT sts:AssumeRoleWithWebIdentity Allow
STRINGEQUALS system:serviceaccount:aws-load-balancer-operator:aws-load-balancer-controller-manager
PRINCIPAL arn:aws:iam:<aws_account_number>:oidc-provider/<cluster_oidc_endpoint>
```

where:

<aws_account_number>

Specifies the ARN of the created AWS IAM role for the AWS Load Balancer Operator, such as **arn:aws:iam::777777777777:role/albo-operator**.

3. Download the permission policy for the AWS Load Balancer Operator by running the following command:

```
$ curl -o albo-operator-permission-policy.json
https://raw.githubusercontent.com/openshift/aws-load-balancer-operator/main/hack/operator-permission-policy.json
```

4. Attach the permission policy for the AWS Load Balancer Controller to the IAM role by running the following command:

```
$ aws iam put-role-policy --role-name albo-operator --policy-name perms-policy-albo-operator --policy-document file://albo-operator-permission-policy.json
```

2.3.2. Configuring the ARN role for the AWS Load Balancer Operator

To authorize the AWS Load Balancer Operator, configure the Amazon Resource Name (ARN) role as an environment variable by using the CLI. This ensures the Operator has the necessary permissions to manage resources within the cluster.

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

1. Create the **aws-load-balancer-operator** project by running the following command:

```
$ oc new-project aws-load-balancer-operator
```

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

```
$ cat <<EOF | oc apply -f -
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: aws-load-balancer-operator
  namespace: aws-load-balancer-operator
spec:
  targetNamespaces: []
EOF
```

3. Create the **Subscription** object by running the following command:

```
$ cat <<EOF | oc apply -f -
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: aws-load-balancer-operator
  namespace: aws-load-balancer-operator
spec:
  channel: stable-v1
  name: aws-load-balancer-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  config:
    env:
    - name: ROLEARN
      value: "<albo_role_arn>"
EOF
```

where:

<albo_role_arn>

Specifies the ARN role to be used in the **CredentialsRequest** to provision the AWS credentials for the AWS Load Balancer Operator. An example for **<albo_role_arn>** is **arn:aws:iam::<aws_account_number>:role/albo-operator**.

> **NOTE**
>
> The AWS Load Balancer Operator waits until the secret is created before moving to the **Available** status.

2.3.3. The IAM role for the AWS Load Balancer Controller

To authorize the AWS Load Balancer Controller, configure the **CredentialsRequest** object with a manually provisioned IAM role. This ensures the controller functions correctly by using the specific permissions defined in your manual provisioning process.

You can create the IAM role by using the following options:

- Using the Cloud Credential Operator utility (**ccoctl**) and a predefined **CredentialsRequest** object.
- Using the AWS CLI and predefined AWS manifests.

If your environment does not support the **ccoctl command.ws-short** CLI, use the AWS CLI.

Additional resources

- the Cloud Credential Operator utility (**ccoctl**)

2.3.3.1. Creating an AWS IAM role for the controller by using the Cloud Credential Operator utility

To enable the AWS Load Balancer Controller to interact with subnets and VPCs, create an IAM role by using the Cloud Credential Operator utility (**ccoctl**). This utility ensures the controller has the specific permissions required to manage network resources within the cluster.

Prerequisites

- You must extract and prepare the **ccoctl** binary.

Procedure

1. Download the **CredentialsRequest** custom resource (CR) and store it in a directory by running the following command:

```
$ curl --create-dirs -o <credentials_requests_dir>/controller.yaml
https://raw.githubusercontent.com/openshift/aws-load-balancer-
operator/main/hack/controller/controller-credentials-request.yaml
```

2. Use the **ccoctl** utility to create an AWS IAM role by running the following command:

```
$ ccoctl aws create-iam-roles \
    --name <name> \
    --region=<aws_region> \
    --credentials-requests-dir=<credentials_requests_dir> \
    --identity-provider-arn <oidc_arn>
```

Example output

```
2023/09/12 11:38:57 Role arn:aws:iam::777777777777:role/<name>-aws-load-balancer-
operator-aws-load-balancer-controller created
2023/09/12 11:38:57 Saved credentials configuration to:
/home/user/<credentials_requests_dir>/manifests/aws-load-balancer-operator-aws-load-
balancer-controller-credentials.yaml
2023/09/12 11:38:58 Updated Role policy for Role <name>-aws-load-balancer-operator-aws-
load-balancer-controller created
```

where:

<name>

Specifies the Amazon Resource Name (ARN) for an AWS IAM role that was created for the AWS Load Balancer Controller, such as **arn:aws:iam::777777777777:role/<name>-aws-load-balancer-operator-aws-load-balancer-controller**.

> **NOTE**
>
> The length of an AWS IAM role name must be less than or equal to 12 characters.

2.3.3.2. Creating an AWS IAM role for the controller by using the AWS CLI

To enable the AWS Load Balancer Controller to interact with subnets and Virtual Private Clouds (VPCs), create an IAM role by using the AWS CLI. This ensures the controller has the specific permissions required to manage network resources within the cluster.

Prerequisites

- You must have access to the AWS command-line interface (**aws**).

Procedure

1. Generate a trust policy file using your identity provider by running the following command:

```
$ cat <<EOF > albo-controller-trust-policy.json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Federated": "<oidc_arn>"
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringEquals": {
                    "<cluster_oidc_endpoint>:sub": "system:serviceaccount:aws-load-balancer-
operator:aws-load-balancer-operator-controller-manager"
                }
            }
        }
    ]
}
EOF
```

where:

<oidc_arn>

Specifies the Amazon Resource Name (ARN) of the OIDC identity provider, such as **arn:aws:iam::777777777777:oidc-provider/rh-oidc.s3.us-east-1.amazonaws.com/28292va7ad7mr9r4he1fb09b14t59t4f**.

serviceaccount

Specifies the service account for the AWS Load Balancer Controller. An example of **<cluster_oidc_endpoint>** is **rh-oidc.s3.us-east-1.amazonaws.com/28292va7ad7mr9r4he1fb09b14t59t4f**.

2. Create an AWS IAM role with the generated trust policy by running the following command:

```
$ aws iam create-role --role-name albo-controller --assume-role-policy-document file://albo-
controller-trust-policy.json
```

Example output

```
ROLE arn:aws:iam::<aws_account_number>:role/albo-controller 2023-08-02T12:13:22Z (1)
ASSUMEROLEPOLICYDOCUMENT 2012-10-17
```

```
STATEMENT sts:AssumeRoleWithWebIdentity Allow
STRINGEQUALS system:serviceaccount:aws-load-balancer-operator:aws-load-balancer-operator-controller-manager
PRINCIPAL arn:aws:iam:<aws_account_number>:oidc-provider/<cluster_oidc_endpoint>
```

where:

<aws_account_number>

Specifies the ARN for an AWS IAM role for the AWS Load Balancer Controller, such as **arn:aws:iam::777777777777:role/albo-controller**.

3. Download the permission policy for the AWS Load Balancer Controller by running the following command:

```
$ curl -o albo-controller-permission-policy.json
https://raw.githubusercontent.com/openshift/aws-load-balancer-operator/main/assets/iam-policy.json
```

4. Attach the permission policy for the AWS Load Balancer Controller to an AWS IAM role by running the following command:

```
$ aws iam put-role-policy --role-name albo-controller --policy-name perms-policy-albo-controller --policy-document file://albo-controller-permission-policy.json
```

5. Create a YAML file that defines the **AWSLoadBalancerController** object:

Example sample-aws-lb-manual-creds.yaml file

```
apiVersion: networking.olm.openshift.io/v1
kind: AWSLoadBalancerController
metadata:
  name: cluster
spec:
  credentialsRequestConfig:
    stsIAMRoleARN: <albc_role_arn>
```

where:

kind

Specifies the **AWSLoadBalancerController** object.

metatdata.name

Specifies the AWS Load Balancer Controller name. All related resources use this instance name as a suffix.

stsIAMRoleARN

Specifies the ARN role for the AWS Load Balancer Controller. The **CredentialsRequest** object uses this ARN role to provision the AWS credentials. An example of **<albc_role_arn>** is **arn:aws:iam::777777777777:role/albo-controller**.

2.3.4. Additional resources

- Configuring the Cloud Credential Operator utility

2.4. INSTALLING THE AWS LOAD BALANCER OPERATOR

The AWS Load Balancer Operator deploys and manages the AWS Load Balancer Controller. You can install the AWS Load Balancer Operator from the software catalog by using OpenShift Container Platform web console or CLI.

2.4.1. Installing the AWS Load Balancer Operator by using the web console

To deploy the AWS Load Balancer Operator, install the Operator by using the web console. You can manage the lifecycle of the Operator by using a graphical interface.

Prerequisites

- You have logged in to the OpenShift Container Platform web console as a user with **cluster-admin** permissions.
- Your cluster is configured with AWS as the platform type and cloud provider.
- If you are using a security token service (STS) or user-provisioned infrastructure, follow the related preparation steps. For example, if you are using AWS Security Token Service, see "Preparing for the AWS Load Balancer Operator on a cluster using the AWS Security Token Service (STS)".

Procedure

1. Navigate to **Operators → OperatorHub** in the OpenShift Container Platform web console.
2. Select the **AWS Load Balancer Operator**. You can use the **Filter by keyword** text box or the filter list to search for the AWS Load Balancer Operator from the list of Operators.
3. Select the **aws-load-balancer-operator** namespace.
4. On the **Install Operator** page, select the following options:
   a. For the **Update the channel** option, select **stable-v1**.
   b. For the **Installation mode** option, select **All namespaces on the cluster (default)**
   c. For the **Installed Namespace** option, select **aws-load-balancer-operator**. If the **aws-load-balancer-operator** namespace does not exist, it gets created during the Operator installation.
   d. Select **Update approval** as **Automatic** or **Manual**. By default, the **Update approval** is set to **Automatic**. If you select automatic updates, the Operator Lifecycle Manager (OLM) automatically upgrades the running instance of your Operator without any intervention. If you select manual updates, the OLM creates an update request. As a cluster administrator, you must then manually approve that update request to have the Operator update to the newer version.
5. Click **Install**.

Verification

- Verify that the AWS Load Balancer Operator shows the **Status** as **Succeeded** on the Installed Operators dashboard.

2.4.2. Installing the AWS Load Balancer Operator by using the CLI

To deploy the AWS Load Balancer Controller, install the AWS Load Balancer Operator by using the command-line interface (CLI).

Prerequisites

- You are logged in to the OpenShift Container Platform web console as a user with **cluster-admin** permissions.
- Your cluster is configured with AWS as the platform type and cloud provider.
- You have logged into the OpenShift CLI (**oc**).

Procedure

1. Create a **Namespace** object:
    a. Create a YAML file that defines the **Namespace** object:

    Example namespace.yaml file

    ```
    apiVersion: v1
    kind: Namespace
    metadata:
      name: aws-load-balancer-operator
    # ...
    ```

    b. Create the **Namespace** object by running the following command:

    ```
    $ oc apply -f namespace.yaml
    ```

2. Create an **OperatorGroup** object:
    a. Create a YAML file that defines the **OperatorGroup** object:

    Example operatorgroup.yaml file

    ```
    apiVersion: operators.coreos.com/v1
    kind: OperatorGroup
    metadata:
      name: aws-lb-operatorgroup
      namespace: aws-load-balancer-operator
    spec:
      upgradeStrategy: Default
    ```

    b. Create the **OperatorGroup** object by running the following command:

    ```
    $ oc apply -f operatorgroup.yaml
    ```

3. Create a **Subscription** object:
    a. Create a YAML file that defines the **Subscription** object:

Example subscription.yaml file

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: aws-load-balancer-operator
  namespace: aws-load-balancer-operator
spec:
  channel: stable-v1
  installPlanApproval: Automatic
  name: aws-load-balancer-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
```

b. Create the **Subscription** object by running the following command:

```
$ oc apply -f subscription.yaml
```

Verification

1. Get the name of the install plan from the subscription:

```
$ oc -n aws-load-balancer-operator \
  get subscription aws-load-balancer-operator \
  --template='{{.status.installplan.name}}{{"\n"}}'
```

2. Check the status of the install plan:

```
$ oc -n aws-load-balancer-operator \
  get ip <install_plan_name> \
  --template='{{.status.phase}}{{"\n"}}'
```

The output must be **Complete**.

2.4.3. Creating the AWS Load Balancer Controller

You can install only a single instance of the **AWSLoadBalancerController** object in a cluster. You can create the AWS Load Balancer Controller by using CLI. The AWS Load Balancer Operator reconciles only the **cluster** named resource.

Prerequisites

- You have created the **echoserver** namespace.
- You have access to the OpenShift CLI (**oc**).

Procedure

1. Create a YAML file that defines the **AWSLoadBalancerController** object:

Example sample-aws-lb.yaml file

```
apiVersion: networking.olm.openshift.io/v1
kind: AWSLoadBalancerController
metadata:
  name: cluster
spec:
  subnetTagging: Auto
  additionalResourceTags:
  - key: example.org/security-scope
    value: staging
  ingressClass: alb
  config:
    replicas: 2
  enabledAddons:
    - AWSWAFv2
```

where:

kind

Specifies the **AWSLoadBalancerController** object.

metadata.name

Specifies the AWS Load Balancer Controller name. The Operator adds this instance name as a suffix to all related resources.

spec.subnetTagging

Specifies the subnet tagging method for the AWS Load Balancer Controller. The following values are valid:

- **Auto**: The AWS Load Balancer Operator determines the subnets that belong to the cluster and tags them appropriately. The Operator cannot determine the role correctly if the internal subnet tags are not present on internal subnet.

- **Manual**: You manually tag the subnets that belong to the cluster with the appropriate role tags. Use this option if you installed your cluster on user-provided infrastructure.

spec.additionalResourceTags

Specifies the tags used by the AWS Load Balancer Controller when it provisions AWS resources.

ingressClass

Specifies the ingress class name. The default value is **alb**.

config.replicas

Specifies the number of replicas of the AWS Load Balancer Controller.

enabledAddons

Specifies annotations as an add-on for the AWS Load Balancer Controller.

AWSWAFv2

Specifies that enablement of the **alb.ingress.kubernetes.io/wafv2-acl-arn** annotation.

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

```
$ oc create -f sample-aws-lb.yaml
```

3. Create a YAML file that defines the **Deployment** resource:

Example sample-aws-lb.yaml file

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: <echoserver>
  namespace: echoserver
spec:
  selector:
    matchLabels:
      app: echoserver
  replicas: 3
  template:
    metadata:
      labels:
        app: echoserver
    spec:
      containers:
        - image: openshift/origin-node
          command:
           - "/bin/socat"
          args:
            - TCP4-LISTEN:8080,reuseaddr,fork
            - EXEC:'/bin/bash -c \"printf \\\"HTTP/1.0 200 OK\r\n\r\n\\\"; sed -e \\\"/^\r/q\\\"\"'
          imagePullPolicy: Always
          name: echoserver
          ports:
            - containerPort: 8080
```

where:

kind

Specifies the deployment resource.

metadata.name

Specifies the deployment name.

spec.replicas

Specifies the number of replicas of the deployment.

4. Create a YAML file that defines the **Service** resource:

Example service-albo.yaml file

```
apiVersion: v1
kind: Service
metadata:
  name: <echoserver>
  namespace: echoserver
spec:
  ports:
    - port: 80
      targetPort: 8080
```

```
    protocol: TCP
  type: NodePort
  selector:
    app: echoserver
```

where:

apiVersion

Specifies the service resource.

metadata.name

Specifies the service name.

5. Create a YAML file that defines the **Ingress** resource:

Example **ingress-albo.yaml** file

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: <name>
  namespace: echoserver
  annotations:
    alb.ingress.kubernetes.io/scheme: internet-facing
    alb.ingress.kubernetes.io/target-type: instance
spec:
  ingressClassName: alb
  rules:
    - http:
        paths:
          - path: /
            pathType: Exact
            backend:
              service:
                name: <echoserver>
                port:
                  number: 80
```

where:

metadata.name

Specifies a name for the **Ingress** resource.

service.name

Specifies the service name.

Verification

- Save the status of the **Ingress** resource in the **HOST** variable by running the following command:

```
$ HOST=$(oc get ingress -n echoserver echoserver --template='{{(index
.status.loadBalancer.ingress 0).hostname}}')
```

- Verify the status of the **Ingress** resource by running the following command:

```
$ curl $HOST
```

2.5. CONFIGURING THE AWS LOAD BALANCER OPERATOR

To automate the provisioning of AWS Load Balancers for your applications, configure the AWS Load Balancer Operator. This setup ensures that the Operator correctly manages ingress resources and external access to your cluster.

2.5.1. Trusting the certificate authority of the cluster-wide proxy

You can configure the cluster-wide proxy in the AWS Load Balancer Operator. After configuring the cluster-wide proxy, Operator Lifecycle Manager (OLM) automatically updates all the deployments of the Operators with the environment variables.

Environment variables include **HTTP_PROXY**, **HTTPS_PROXY**, and **NO_PROXY**. These variables are populated to the managed controller by the AWS Load Balancer Operator.

Procedure

1. Create the config map to contain the certificate authority (CA) bundle in the **aws-load-balancer-operator** namespace by running the following command:

```
$ oc -n aws-load-balancer-operator create configmap trusted-ca
```

2. To inject the trusted CA bundle into the config map, add the **config.openshift.io/inject-trusted-cabundle=true** label to the config map by running the following command:

```
$ oc -n aws-load-balancer-operator label cm trusted-ca config.openshift.io/inject-trusted-cabundle=true
```

3. Update the AWS Load Balancer Operator subscription to access the config map in the AWS Load Balancer Operator deployment by running the following command:

```
$ oc -n aws-load-balancer-operator patch subscription aws-load-balancer-operator --type='merge' -p '{"spec":{"config":{"env":[{"name":"TRUSTED_CA_CONFIGMAP_NAME","value":"trusted-ca"}],"volumes":[{"name":"trusted-ca","configMap":{"name":"trusted-ca"}}],"volumeMounts":[{"name":"trusted-ca","mountPath":"/etc/pki/tls/certs/albo-tls-ca-bundle.crt","subPath":"ca-bundle.crt"}]}}}'
```

4. After the AWS Load Balancer Operator is deployed, verify that the CA bundle is added to the **aws-load-balancer-operator-controller-manager** deployment by running the following command:

```
$ oc -n aws-load-balancer-operator exec deploy/aws-load-balancer-operator-controller-manager -c manager -- bash -c "ls -l /etc/pki/tls/certs/albo-tls-ca-bundle.crt; printenv TRUSTED_CA_CONFIGMAP_NAME"
```

Example output

```
-rw-r--r--. 1 root 1000690000 5875 Jan 11 12:25 /etc/pki/tls/certs/albo-tls-ca-bundle.crt
trusted-ca
```

5. Optional: Restart deployment of the AWS Load Balancer Operator every time the config map changes by running the following command:

```
$ oc -n aws-load-balancer-operator rollout restart deployment/aws-load-balancer-operator-controller-manager
```

Additional resources

- Certificate injection using Operators

2.5.2. Adding TLS termination on the AWS Load Balancer

To secure traffic for your domain, configure TLS termination on the AWS Load Balancer. This setup routes traffic to the pods of a service while ensuring that encrypted connections are decrypted at the load balancer level.

Prerequisites

- You have access to the OpenShift CLI (**oc**).

Procedure

1. Create a YAML file that defines the **AWSLoadBalancerController** resource:

   Example add-tls-termination-albc.yaml file

```
apiVersion: networking.olm.openshift.io/v1
kind: AWSLoadBalancerController
metadata:
  name: cluster
spec:
  subnetTagging: Auto
  ingressClass: tls-termination
# ...
```

   where:

   spec.ingressClass

   Specifies the ingress class name. If the ingress class is not present in your cluster the AWS Load Balancer Controller creates one. The AWS Load Balancer Controller reconciles the additional ingress class values if **spec.controller** is set to **ingress.k8s.aws/alb**.

2. Create a YAML file that defines the **Ingress** resource:

   Example add-tls-termination-ingress.yaml file

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
```

```
  name: <example>
  annotations:
    alb.ingress.kubernetes.io/scheme: internet-facing
    alb.ingress.kubernetes.io/certificate-arn: arn:aws:acm:us-west-2:xxxxx
spec:
  ingressClassName: tls-termination
  rules:
  - host: example.com
    http:
        paths:
          - path: /
            pathType: Exact
            backend:
              service:
                name: <example_service>
                port:
                  number: 80
# ...
```

where:

metadata.name
: Specifies the ingress name.

annotations.alb.ingress.kubernetes.io/scheme
: Specifies the controller that provisions the load balancer for ingress. The provisioning happens in a public subnet to access the load balancer over the internet.

annotations.alb.ingress.kubernetes.io/certificate-arn
: Specifies the Amazon Resource Name (ARN) of the certificate that you attach to the load balancer.

spec.ingressClassName
: Specifies the ingress class name.

rules.host
: Specifies the domain for traffic routing.

backend.service
: Specifies the service for traffic routing.

2.5.3. Creating multiple ingress resources through a single AWS Load Balancer

To route traffic to different services within a single domain, configure multiple ingress resources on a single AWS Load Balancer. This setup allows each resource to provide different endpoints while sharing the same load balancing infrastructure.

Prerequisites

- You have access to the OpenShift CLI (**oc**).

Procedure

1. Create an **IngressClassParams** resource YAML file, for example, **sample-single-lb-params.yaml**, as follows:

```
apiVersion: elbv2.k8s.aws/v1beta1
kind: IngressClassParams
metadata:
  name: single-lb-params
spec:
  group:
    name: single-lb
```

where:

apiVersion

Specifies the API group and version of the **IngressClassParams** resource.

metadata.name

Specifies the **IngressClassParams** resource name.

spec.group.name

Specifies the **IngressGroup** resource name. All of the **Ingress** resources of this class belong to this **IngressGroup**.

2. Create the **IngressClassParams** resource by running the following command:

```
$ oc create -f sample-single-lb-params.yaml
```

3. Create the **IngressClass** resource YAML file, for example, **sample-single-lb-class.yaml**, as follows:

```
apiVersion: networking.k8s.io/v1
kind: IngressClass
metadata:
  name: single-lb
spec:
  controller: ingress.k8s.aws/alb
  parameters:
    apiGroup: elbv2.k8s.aws
    kind: IngressClassParams
    name: single-lb-params
```

where:

apiVersion

Specifies the API group and version of the **IngressClass** resource.

metadata.name

Specifies the ingress class name.

spec.controller

Specifies the controller name. The **ingress.k8s.aws/alb** value denotes that all ingress resources of this class should be managed by the AWS Load Balancer Controller.

parameters.apiGroup

Specifies the API group of the **IngressClassParams** resource.

parameters.kind

Specifies the resource type of the **IngressClassParams** resource.

parameters.name

Specifies the **IngressClassParams** resource name.

4. Create the **IngressClass** resource by running the following command:

```
$ oc create -f sample-single-lb-class.yaml
```

5. Create the **AWSLoadBalancerController** resource YAML file, for example, **sample-single-lb.yaml**, as follows:

```
apiVersion: networking.olm.openshift.io/v1
kind: AWSLoadBalancerController
metadata:
  name: cluster
spec:
  subnetTagging: Auto
  ingressClass: single-lb
```

where:

spec.ingressClass

Specifies the name of the **IngressClass** resource.

6. Create the **AWSLoadBalancerController** resource by running the following command:

```
$ oc create -f sample-single-lb.yaml
```

7. Create the **Ingress** resource YAML file, for example, **sample-multiple-ingress.yaml**, as follows:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: example-1
  annotations:
    alb.ingress.kubernetes.io/scheme: internet-facing
    alb.ingress.kubernetes.io/group.order: "1"
    alb.ingress.kubernetes.io/target-type: instance
spec:
  ingressClassName: single-lb
  rules:
  - host: example.com
    http:
        paths:
        - path: /blog
          pathType: Prefix
          backend:
            service:
              name: example-1
              port:
                number: 80
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
```

```
  name: example-2
  annotations:
    alb.ingress.kubernetes.io/scheme: internet-facing
    alb.ingress.kubernetes.io/group.order: "2"
    alb.ingress.kubernetes.io/target-type: instance
spec:
  ingressClassName: single-lb
  rules:
  - host: example.com
    http:
        paths:
        - path: /store
          pathType: Prefix
          backend:
            service:
              name: example-2
              port:
                number: 80
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: example-3
  annotations:
    alb.ingress.kubernetes.io/scheme: internet-facing
    alb.ingress.kubernetes.io/group.order: "3"
    alb.ingress.kubernetes.io/target-type: instance
spec:
  ingressClassName: single-lb
  rules:
  - host: example.com
    http:
        paths:
        - path: /
          pathType: Prefix
          backend:
            service:
              name: example-3
              port:
                number: 80
```

where:

metadata.name

Specifies the ingress name.

alb.ingress.kubernetes.io/scheme

Specifies the load balancer to provision in the public subnet to access the internet.

alb.ingress.kubernetes.io/group.order

Specifies the order in which the rules from the multiple ingress resources are matched when the request is received at the load balancer.

alb.ingress.kubernetes.io/target-type

Specifies that the load balancer will target OpenShift Container Platform nodes to reach the service.

spec.ingressClassName
Specifies the ingress class that belongs to this ingress.

rules.host
Specifies a domain name used for request routing.

http.paths.path
Specifies the path that must route to the service.

backend.service.name
Specifies the service name that serves the endpoint configured in the **Ingress** resource.

port.number
Specifies the port on the service that serves the endpoint.

8. Create the **Ingress** resource by running the following command:

```
$ oc create -f sample-multiple-ingress.yaml
```

2.5.4. AWS Load Balancer Operator logs

To troubleshoot the AWS Load Balancer Operator, view the logs using the **oc logs** command. By viewing the logs, you can diagnose issues and monitor the activity of the Operator.

Procedure

- View the logs of the AWS Load Balancer Operator by running the following command:

```
$ oc logs -n aws-load-balancer-operator deployment/aws-load-balancer-operator-controller-manager -c manager
```

CHAPTER 3. EBPF MANAGER OPERATOR

3.1. ABOUT THE EBPF MANAGER OPERATOR

You can use the eBPF Manager Operator to centralize and secure the deployment of eBPF programs in a Kubernetes cluster. The eBPF Manager Operator streamlines lifecycle management and provides system-wide visibility so that you can focus on program interaction rather than manual configuration.

IMPORTANT

eBPF Manager Operator 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 Technology Preview Features Support Scope .

3.1.1. About Extended Berkeley Packet Filter (eBPF)

eBPF extends the original Berkeley Packet Filter for advanced network traffic filtering. It acts as a virtual machine inside the Linux kernel, allowing you to run sandboxed programs in response to events such as network packets, system calls, or kernel functions.

3.1.2. About the eBPF Manager Operator

eBPF Manager simplifies the management and deployment of eBPF programs within Kubernetes, as well as enhancing the security around using eBPF programs. It utilizes Kubernetes custom resource definitions (CRDs) to manage eBPF programs packaged as OCI container images. This approach helps to delineate deployment permissions and enhance security by restricting program types deployable by specific users.

eBPF Manager is a software stack designed to manage eBPF programs within Kubernetes. It facilitates the loading, unloading, modifying, and monitoring of eBPF programs in Kubernetes clusters. It includes a daemon, CRDs, an agent, and an operator:

bpfman

A system daemon that manages eBPF programs via a gRPC API.

eBPF CRDs

A set of CRDs like XdpProgram and TcProgram for loading eBPF programs, and a bpfman-generated CRD (BpfProgram) for representing the state of loaded programs.

bpfman-agent

Runs within a daemonset container, ensuring eBPF programs on each node are in the desired state.

bpfman-operator

Manages the lifecycle of the bpfman-agent and CRDs in the cluster using the Operator SDK.

The eBPF Manager Operator offers the following features:

- Enhances security by centralizing eBPF program loading through a controlled daemon. eBPF Manager has the elevated privileges so the applications don't need to be. eBPF program control is regulated by standard Kubernetes Role-based access control (RBAC), which can allow or deny an application's access to the different eBPF Manager CRDs that manage eBPF program loading and unloading.

- Provides detailed visibility into active eBPF programs, improving your ability to debug issues across the system.
- Facilitates the coexistence of multiple eBPF programs from different sources using protocols like libxdp for XDP and TC programs, enhancing interoperability.
- Streamlines the deployment and lifecycle management of eBPF programs in Kubernetes. Developers can focus on program interaction rather than lifecycle management, with support for existing eBPF libraries like Cilium, libbpf, and Aya.

3.1.3. Additional resources

- eBPF Documentation
- bpfman
- eBPF Manager custom resource definition (CRD) API specification

3.2. INSTALLING THE EBPF MANAGER OPERATOR

To manage eBPF programs across your cluster nodes, you can install the eBPF Manager Operator by using the OpenShift Container Platform CLI or the web console. This Operator provides a standardized way to deploy, monitor, and secure eBPF-based networking and observability tools.

IMPORTANT

eBPF Manager Operator 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 Technology Preview Features Support Scope .

3.2.1. Installing the eBPF Manager Operator using the CLI

To manage eBPF programs across your cluster nodes, you can install the eBPF Manager Operator by using the OpenShift Container Platform CLI. This process involves creating a dedicated namespace and subscribing to the Operator to enable node-level networking and observability tools.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have an account with administrator privileges.

Procedure

1. To create the **bpfman** namespace, enter the following command:

```
$ cat << EOF| oc create -f -
apiVersion: v1
kind: Namespace
metadata:
  labels:
    pod-security.kubernetes.io/enforce: privileged
    pod-security.kubernetes.io/enforce-version: v1.24
  name: bpfman
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: bpfman-operators
  namespace: bpfman
EOF
```

3. Subscribe to the eBPF Manager Operator.

   a. To create a **Subscription** CR for the eBPF Manager Operator, enter the following command:

```
$ cat << EOF| oc create -f -
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: bpfman-operator
  namespace: bpfman
spec:
  name: bpfman-operator
  channel: alpha
  source: community-operators
  sourceNamespace: openshift-marketplace
EOF
```

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

```
$ oc get ip -n bpfman
```

Example output

```
NAME          CSV                                APPROVAL    APPROVED
install-ppjxl   security-profiles-operator.v0.8.5   Automatic   true
```

5. To verify the version of the Operator, enter the following command:

```
$ oc get csv -n bpfman
```

Example output

```
NAME                        DISPLAY                 VERSION   REPLACES
PHASE
bpfman-operator.v0.5.0            eBPF Manager Operator            0.5.0     bpfman-
operator.v0.4.2            Succeeded
```

3.2.2. Installing the eBPF Manager Operator using the web console

To manage eBPF programs across your cluster nodes, you can install the eBPF Manager Operator by using the OpenShift Container Platform web console. You can use the eBPF Manager Operator to enable node-level networking and observability tools through the OperatorHub interface.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have an account with administrator privileges.

Procedure

1. Install the eBPF Manager Operator:
    a. In the OpenShift Container Platform web console, click **Operators → OperatorHub**.
    b. Select **eBPF Manager Operator** from the list of available Operators, and if prompted to **Show community Operator**, click **Continue**.
    c. Click **Install**.
    d. On the **Install Operator** page, under **Installed Namespace**, select **Operator recommended Namespace**.
    e. Click **Install**.
2. Verify that the eBPF Manager Operator is installed successfully:
    a. Navigate to the **Operators → Installed Operators** page.
    b. Ensure that **eBPF Manager Operator** is listed in the **openshift-ingress-node-firewall** project with a **Status** of **InstallSucceeded**.

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

    If the Operator does not have a **Status** of **InstallSucceeded**, troubleshoot using the following steps:

    - Inspect the **Operator Subscriptions** and **Install Plans** tabs for any failures or errors under **Status**.
    - Navigate to the **Workloads → Pods** page and check the logs for pods in the **bpfman** project.

Additional resources

- Deploying a containerized eBPF program
- Configuring Ingress Node Firewall Operator to use the eBPF Manager Operator

3.3. DEPLOYING AN EBPF PROGRAM

As a cluster administrator, you can deploy containerized eBPF applications by using the eBPF Manager Operator. This process involves loading an eBPF program through a custom resource and deploying a user-space daemon set that accesses eBPF maps without requiring privileged permissions.

As a cluster administrator, you can deploy containerized eBPF applications with the eBPF Manager Operator.

For the example eBPF program deployed in this procedure, the sample manifest does the following:

First, it creates basic Kubernetes objects like **Namespace**, **ServiceAccount**, and **ClusterRoleBinding**. It also creates a **XdpProgram** object, which is a custom resource definition (CRD) that eBPF Manager provides, that loads the eBPF XDP program. Each program type has it's own CRD, but they are similar in what they do. For more information, see Loading eBPF Programs On Kubernetes .

Second, it creates a daemon set which runs a user space program that reads the eBPF maps that the eBPF program is populating. This eBPF map is volume mounted using a Container Storage Interface (CSI) driver. By volume mounting the eBPF map in the container in lieu of accessing it on the host, the application pod can access the eBPF maps without being privileged. For more information on how the CSI is configured, see See Deploying an eBPF enabled application On Kubernetes .

> **IMPORTANT**
>
> eBPF Manager Operator 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 Technology Preview Features Support Scope .

3.3.1. Deploying a containerized eBPF program

To run custom networking or security logic on your cluster nodes, you can deploy containerized eBPF programs. You can use containerized eBPF programs to monitor kernel events and manage network traffic efficiently at the node level.

As a cluster administrator, you can deploy an eBPF program to nodes on your cluster. In this procedure, a sample containerized eBPF program is installed in the **go-xdp-counter** namespace.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have an account with administrator privileges.
- You have installed the eBPF Manager Operator.

Procedure

1. To download the manifest, enter the following command:

```
$ curl -L https://github.com/bpfman/bpfman/releases/download/v0.5.1/go-xdp-counter-install-selinux.yaml -o go-xdp-counter-install-selinux.yaml
```

2. To deploy the sample eBPF application, enter the following command:

```
$ oc create -f go-xdp-counter-install-selinux.yaml
```

Example output

```
namespace/go-xdp-counter created
serviceaccount/bpfman-app-go-xdp-counter created
clusterrolebinding.rbac.authorization.k8s.io/xdp-binding created
daemonset.apps/go-xdp-counter-ds created
xdpprogram.bpfman.io/go-xdp-counter-example created
selinuxprofile.security-profiles-operator.x-k8s.io/bpfman-secure created
```

3. To confirm that the eBPF sample application deployed successfully, enter the following command:

```
$ oc get all -o wide -n go-xdp-counter
```

Example output

```
NAME                          READY   STATUS    RESTARTS   AGE   IP            NODE
NOMINATED NODE   READINESS GATES
pod/go-xdp-counter-ds-4m9cw   1/1     Running   0          44s   10.129.0.92    ci-ln-dcbq7d2-
72292-ztrkp-master-1   <none>           <none>
pod/go-xdp-counter-ds-7hzww   1/1     Running   0          44s   10.130.0.86    ci-ln-dcbq7d2-
72292-ztrkp-master-2   <none>           <none>
pod/go-xdp-counter-ds-qm9zx   1/1     Running   0          44s   10.128.0.101   ci-ln-dcbq7d2-
72292-ztrkp-master-0   <none>           <none>

NAME                               DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE
NODE SELECTOR   AGE   CONTAINERS       IMAGES
SELECTOR
daemonset.apps/go-xdp-counter-ds   3         3         3       3            3           <none>          44s
go-xdp-counter   quay.io/bpfman-userspace/go-xdp-counter:v0.5.0   name=go-xdp-counter
```

4. To confirm that the example XDP program is running, enter the following command:

```
$ oc get xdpprogram go-xdp-counter-example
```

Example output

```
NAME                     BPFFUNCTIONNAME   NODESELECTOR   STATUS
go-xdp-counter-example   xdp_stats         {}             ReconcileSuccess
```

5. To confirm that the XDP program is collecting data, enter the following command:

```
$ oc logs <pod_name> -n go-xdp-counter
```

Replace **<pod_name>** with the name of an XDP program pod, such as **go-xdp-counter-ds-4m9cw**.

Example output

```
2024/08/13 15:20:06 15016 packets received
2024/08/13 15:20:06 93581579 bytes received
...
```

CHAPTER 4. EXTERNAL DNS OPERATOR

4.1. EXTERNAL DNS OPERATOR RELEASE NOTES

The External DNS Operator deploys and manages **ExternalDNS** to provide name resolution for services and routes. This enables your external DNS provider to resolve hostnames directly to OpenShift Container Platform resources.

IMPORTANT

The External DNS Operator is only supported on the **x86_64** architecture.

These release notes track the development of the External DNS Operator in OpenShift Container Platform.

4.1.1. External DNS Operator 1.3

The External DNS Operator 1.3 release notes summarize all new features and enhancements, notable technical changes, major corrections from previous versions, and any known bugs upon general availability.

External DNS Operator 1.3.2

The following advisory is available for the External DNS Operator version 1.3.2:

- RHEA-2025:22454 Product Enhancement Advisory

External DNS Operator 1.3.1

The following advisory is available for the External DNS Operator version 1.3.1:

- RHEA-2025:15598 Product Enhancement Advisory
  This update includes improved container security.

External DNS Operator 1.3.0

The following advisory is available for the External DNS Operator version 1.3.0:

- RHEA-2024:8550 Product Enhancement Advisory
  This update includes a rebase to the 0.14.2 version of the upstream project.

  Bug fixes:

- Previously, the ExternalDNS Operator could not deploy operands on HCP clusters. With this release, the Operator deploys operands in a running and ready state. (OCPBUGS-37059)

- Previously, the ExternalDNS Operator was not using RHEL 9 as its building or base images. With this release, RHEL9 is the base. (OCPBUGS-41683)

- Previously, the godoc had a broken link for Infoblox provider. With this release, the godoc is revised for accuracy. Some links are removed while some other are replaced with GitHub permalinks. (OCPBUGS-36797)

4.1.2. External DNS Operator 1.2

The External DNS Operator 1.2 release notes summarize all new features and enhancements, notable technical changes, major corrections from previous versions, and any known bugs upon general availability.

External DNS Operator 1.2.0

The following advisory is available for the External DNS Operator version 1.2.0:

- RHEA-2022:5867 ExternalDNS Operator 1.2 Operator or operand containers
  New features:
- The External DNS Operator now supports AWS shared VPC. For more information, see "Creating DNS records in a different AWS Account using a shared VPC".
  Bug fixes:
- The update strategy for the operand changed from **Rolling** to **Recreate**. (OCPBUGS-3630)

Additional resources

- Creating DNS records in a different AWS Account using a shared VPC

4.1.3. External DNS Operator 1.1

The External DNS Operator 1.1 release notes summarize all new features and enhancements, notable technical changes, major corrections from previous versions, and any known bugs upon general availability.

External DNS Operator 1.1.1

The following advisory is available for the External DNS Operator version 1.1.1:

- RHEA-2024:0536 ExternalDNS Operator 1.1 Operator or operand containers

External DNS Operator 1.1.0

This release included a rebase of the operand from the upstream project version 0.13.1. The following advisory is available for the External DNS Operator version 1.1.0:

- RHEA-2022:9086-01 ExternalDNS Operator 1.1 Operator or operand containers
  Bug fixes:
- Previously, the ExternalDNS Operator enforced an empty **defaultMode** value for volumes, which caused constant updates due to a conflict with the OpenShift API. Now, the **defaultMode** value is not enforced and operand deployment does not update constantly. (OCPBUGS-2793)

4.1.4. External DNS Operator 1.0

The External DNS Operator 1.0 release notes summarize all new features and enhancements, notable technical changes, major corrections from previous versions, and any known bugs upon general availability.

External DNS Operator 1.0.1

The following advisory is available for the External DNS Operator version 1.0.1:

- RHEA-2024:0537 ExternalDNS Operator 1.0 Operator or operand containers

External DNS Operator 1.0.0

The following advisory is available for the External DNS Operator version 1.0.0:

- RHEA-2022:5867 ExternalDNS Operator 1.0 Operator operand containers
  Bug fixes:

- Previously, the External DNS Operator issued a warning about the violation of the restricted SCC policy during ExternalDNS operand pod deployments. This issue has been resolved. (BZ#2086408)

4.2. UNDERSTANDING THE EXTERNAL DNS OPERATOR

To provide name resolution for services and routes from an External DNS provider to OpenShift Container Platform, use the External DNS Operator. This Operator deploys and manages **ExternalDNS** to synchronize your cluster resources with the external provider.

4.2.1. External DNS Operator domain name limitations

To prevent configuration errors when deploying the **ExternalDNS** resource, review the domain name limitations enforced by the External DNS Operator. Understanding these constraints ensures that your requested hostnames and domains are compatible with your underlying DNS provider.

The External DNS Operator uses the TXT registry that adds the prefix for TXT records. This reduces the maximum length of the domain name for TXT records. A DNS record cannot be present without a corresponding TXT record, so the domain name of the DNS record must follow the same limit as the TXT records. For example, a DNS record of **<domain_name_from_source>** results in a TXT record of **external-dns-<record_type>-<domain_name_from_source>**.

The domain name of the DNS records generated by the External DNS Operator has the following limitations:

Record type	Number of characters
CNAME	44
Wildcard CNAME records on AzureDNS	42
A	48
Wildcard A records on AzureDNS	46

The following error shows in the External DNS Operator logs if the generated domain name exceeds any of the domain name limitations:

```
time="2022-09-02T08:53:57Z" level=error msg="Failure in zone test.example.io. [Id:
/hostedzone/Z06988883Q0H0RL6UMXXX]"
time="2022-09-02T08:53:57Z" level=error msg="InvalidChangeBatch: [FATAL problem:
DomainLabelTooLong (Domain label is too long) encountered with 'external-dns-a-hello-openshift-
aaaaaaaaaa-bbbbbbbbbb-ccccccc']\n\tstatus code: 400, request id: e54dfd5a-06c6-47b0-bcb9-
a4f7c3a4e0c6"
```

4.2.2. Deploying the External DNS Operator

You can deploy the External DNS Operator on-demand from the Software Catalog. Deploying the External DNS Operator creates a **Subscription** object.

The External DNS Operator implements the External DNS API from the **olm.openshift.io** API group. The External DNS Operator updates services, routes, and external DNS providers.

Prerequisites

- You have installed the **yq** CLI tool.

Procedure

1. Check the name of an install plan, such as **install-zcvlr**, by running the following command:

```
$ oc -n external-dns-operator get sub external-dns-operator -o yaml | yq
'.status.installplan.name'
```

2. Check if the status of an install plan is **Complete** by running the following command:

```
$ oc -n external-dns-operator get ip <install_plan_name> -o yaml | yq '.status.phase'
```

3. View the status of the **external-dns-operator** deployment by running the following command:

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

Example output

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

4.2.3. Viewing External DNS Operator logs

To troubleshoot DNS configuration issues, view the External DNS Operator logs. Use the **oc logs** command to retrieve diagnostic information directly from the Operator pod.

Procedure

- View the logs of the External DNS Operator by running the following command:

```
$ oc logs -n external-dns-operator deployment/external-dns-operator -c external-dns-operator
```

4.3. INSTALLING THE EXTERNAL DNS OPERATOR

To manage DNS records on your cloud infrastructure, install the External DNS Operator. This Operator supports deployment on major cloud providers, including Amazon Web Services (AWS), Microsoft Azure, and Google Cloud.

4.3.1. Installing the External DNS Operator with OperatorHub

You can install the External DNS Operator by using the OpenShift Container Platform OperatorHub. You can then manage the Operator lifecycle directly from the web console.

Procedure

1. Click **Operators → OperatorHub** in the OpenShift Container Platform web console.
2. Click **External DNS Operator**. You can use the **Filter by keyword** text box or the filter list to search for External DNS Operator from the list of Operators.
3. Select the **external-dns-operator** namespace.
4. On the **External DNS Operator** page, click **Install**.
5. On the **Install Operator** page, ensure that you selected the following options:
    a. Update the channel as **stable-v1**.
    b. Installation mode as **A specific name on the cluster**.
    c. Installed namespace as **external-dns-operator**. If namespace **external-dns-operator** does not exist, the Operator gets created during the Operator installation.
    d. Select **Approval Strategy** as **Automatic** or **Manual**. The Approval Strategy defaults to **Automatic**.
    e. Click **Install**.
    If you select **Automatic** updates, the Operator Lifecycle Manager (OLM) automatically upgrades the running instance of your Operator without any intervention.

    If you select **Manual** updates, the OLM creates an update request. As a cluster administrator, you must then manually approve that update request to have the Operator updated to the new version.

Verification

- Verify that the External DNS Operator shows the **Status** as **Succeeded** on the **Installed Operators** dashboard.

4.3.2. Installing the External DNS Operator by using the CLI

You can use the OpenShift CLI (**oc**) to install the External DNS Operator. The Operator manages the installation process directly from your terminal without you having to use the web console.

Prerequisites

- You are logged in to the OpenShift CLI (**oc**).

Procedure

1. Create a **Namespace** object:
    a. Create a YAML file that defines the **Namespace** object:

    Example namespace.yaml file

    -

```
apiVersion: v1
kind: Namespace
metadata:
  name: external-dns-operator
# ...
```

b. Create the **Namespace** object by running the following command:

```
$ oc apply -f namespace.yaml
```

2. Create an **OperatorGroup** object:

a. Create a YAML file that defines the **OperatorGroup** object:

Example **operatorgroup.yaml** file

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: external-dns-operator
  namespace: external-dns-operator
spec:
  upgradeStrategy: Default
  targetNamespaces:
  - external-dns-operator
# ...
```

b. Create the **OperatorGroup** object by running the following command:

```
$ oc apply -f operatorgroup.yaml
```

3. Create a **Subscription** object:

a. Create a YAML file that defines the **Subscription** object:

Example **subscription.yaml** file

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: external-dns-operator
  namespace: external-dns-operator
spec:
  channel: stable-v1
  installPlanApproval: Automatic
  name: external-dns-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
# ...
```

b. Create the **Subscription** object by running the following command:

```
$ oc apply -f subscription.yaml
```

Verification

1. Get the name of the install plan from the subscription by running the following command:

```
$ oc -n external-dns-operator \
  get subscription external-dns-operator \
  --template='{{.status.installplan.name}}{{"\n"}}'
```

2. Verify that the status of the install plan is **Complete** by running the following command:

```
$ oc -n external-dns-operator \
  get ip <install_plan_name> \
  --template='{{.status.phase}}{{"\n"}}'
```

3. Verify that the status of the **external-dns-operator** pod is **Running** by running the following command:

```
$ oc -n external-dns-operator get pod
```

Example output

```
NAME                                     READY   STATUS    RESTARTS   AGE
external-dns-operator-5584585fd7-5lwqm   2/2     Running   0          11m
```

4. Verify that the catalog source of the subscription is **redhat-operators** by running the following command:

```
$ oc -n external-dns-operator get subscription
```

5. Check the **external-dns-operator** version by running the following command:

```
$ oc -n external-dns-operator get csv
```

4.4. EXTERNAL DNS OPERATOR CONFIGURATION PARAMETERS

To customize the behavior of the External DNS Operator, configure the available parameters in the **ExternalDNS** custom resource (CR). By configuraing parameters, you can control how the Operator synchronizes services and routes with your external DNS provider.

4.4.1. External DNS Operator configuration parameters

To customize the behavior of the External DNS Operator, configure the available parameters in the **ExternalDNS** custom resource (CR). By configuring parameters, you can control how the Operator synchronizes services and routes with your external DNS provider.

Parameter	Description

Parameter	Description

spec	Enables the type of a cloud provider.
	
	spec:
	  provider:
	    type: AWS
	    aws:
	      credentials:
	        name: aws-access-key
	
	• **provider.type**: Specifies available options such as AWS, Google Cloud, Azure, and Infoblox.
	• **provider.aws.credentials.name**: Specifies a secret name for your cloud provider.

zones	Enables you to specify DNS zones by their domains. If you do not specify zones, the **ExternalDNS** resource discovers all of the zones present in your cloud provider account. zones: - "<zone_id>" • **<zone_id>**: Specifies the name of DNS zones.

domains	Enables you to specify AWS zones by their domains. If you do not specify domains, the **ExternalDNS** resource discovers all of the zones present in your cloud provider account.
	
	domains:
	- filterType: Include
	  matchType: Exact
	  name: "myzonedomain1.com"
	- filterType: Include
	  matchType: Pattern
	  pattern: ".*\\.otherzonedomain\\.com"
	
	• **domains.filterType**: Specifies that the **ExternalDNS** resource includes the domain name.
	• **domains.matchType**: Specifies that the domain matching has to be exact as opposed to regular expression match.
	• **domains.name**: Specifies the name of the domain.
	• **filterType.matchType**: Specifies the **regex-domain-filter** flag in the **ExternalDNS** resource. You can limit possible domains by using a Regex filter.
	• **filterType.pattern**: Specifies the regex pattern to be used by the **ExternalDNS** resource to filter the domains of the target zones.

<table>
<tr><th>Parameter</th><th>Description</th></tr>
<tr><td>source</td><td>
Enables you to specify the source for the DNS records, Service or Route.
<pre>
source:
  type: Service
  service:
    serviceType:
      - LoadBalancer
      - ClusterIP
  labelFilter:
    matchLabels:
      external-dns.mydomain.org/publish: "yes"
  hostnameAnnotation: "Allow"
  fqdnTemplate:
  - "{{.Name}}.myzonedomain.com"
</pre>

source: Specifies the settings for the source of DNS records.
source.type: Specifies that the ExternalDNS CR uses the Service type as the source for creating DNS records.
service.serviceType: Specifies the service-type-filter flag in the ExternalDNS resource. The serviceType contains the following fields: default: LoadBalancer; expected: ClusterIP; NodePort; LoadBalancer; ExternalName.
service.labelFilter: Specifies that the controller considers only those resources that match with label filter.
hostnameAnnotation: Specifies that the default value for hostnameAnnotation is Ignore which instructs ExternalDNS to generate DNS records by using the templates specified in the field fqdnTemplates. When the value is Allow the DNS records get generated based on the value specified in the external-dns.alpha.kubernetes.io/hostname annotation.
fqdnTemplate: Specifies that the External DNS Operator uses a string to generate DNS names from sources that do not define a hostname, or to add a hostname suffix when paired with the fake source.

<pre>
source:
  type: OpenShiftRoute
  openshiftRouteOptions:
    routerName: default
    labelFilter:
      matchLabels:
        external-dns.mydomain.org/publish: "yes"
</pre>

source.type: Specifies the creation of DNS records.
openshiftRouteOptions.routerName: Specifies if the source type is OpenShiftRoute. If so, you can pass the Ingress Controller name. The ExternalDNS resource uses the canonical name of the Ingress Controller as the target for CNAME records.

</td></tr>
</table>

4.5. CREATING DNS RECORDS ON AWS

To create DNS records on AWS and AWS GovCloud, use the External DNS Operator. The Operator manages external name resolution for your cluster services directly through the Operator.

> **IMPORTANT**
>
> Usage of External DNS Operator on an STS-enabled cluster that runs in AWS Government (AWS GovCloud) regions is not supported.

4.5.1. Creating DNS records on a public hosted zone for AWS by using Red Hat External DNS Operator

You can create DNS records on a public hosted zone for AWS by using the Red Hat External DNS Operator. You can use the same instructions to create DNS records on a hosted zone for AWS GovCloud.

Procedure

1. Check the user profile by running the following command. The profile, such as **system:admin**, must have access to the **kube-system** namespace. If you do not have the credentials, you can fetch the credentials from the **kube-system** namespace to use the cloud provider client by running the following command:

```
$ oc whoami
```

2. Fetch the values from the **aws-creds** secret that exists in the **kube-system** namespace.

```
$ export AWS_ACCESS_KEY_ID=$(oc get secrets aws-creds -n kube-system  --template={{.data.aws_access_key_id}} | base64 -d)
```

```
$ export AWS_SECRET_ACCESS_KEY=$(oc get secrets aws-creds -n kube-system  --template={{.data.aws_secret_access_key}} | base64 -d)
```

3. Get the routes to check the domain:

```
$ oc get routes --all-namespaces | grep console
```

Example output

```
openshift-console          console             console-openshift-console.apps.testextdnsoperator.apacshift.support                       console             https   reencrypt/Redirect     None
openshift-console          downloads           downloads-openshift-console.apps.testextdnsoperator.apacshift.support                    downloads           http    edge/Redirect          None
```

4. Get the list of DNS zones and find the DNS zone that corresponds to the domain of the route that you previously queried:

```
$ aws route53 list-hosted-zones | grep testextdnsoperator.apacshift.support
```

Example output

```
HOSTEDZONES terraform /hostedzone/Z02355203TNN1XXXX1J6O
testextdnsoperator.apacshift.support. 5
```

5. Create the **ExternalDNS** CR for the **route** source:

```
$ cat <<EOF | oc create -f -
apiVersion: externaldns.olm.openshift.io/v1beta1
kind: ExternalDNS
metadata:
  name: sample-aws
spec:
  domains:
  - filterType: Include
    matchType: Exact
    name: testextdnsoperator.apacshift.support
  provider:
    type: AWS
  source:
    type: OpenShiftRoute
    openshiftRouteOptions:
      routerName: default
EOF
```

where:

metadata.name
: Specifies the name of the external DNS resource.

spec.domains
: By default all hosted zones are selected as potential targets. You can include a hosted zone that you need.

domains.matchType
: Specifies that the matching of the domain from the target zone has to be exact. Exact as opposed to regular expression match.

domains.name
: Specifies the exact domain of the zone you want to update. The hostname of the routes must be subdomains of the specified domain.

provider.type
: Specifies the **AWS Route53** DNS provider.

source
: Specifies the options for the source of DNS records.

source.type
: Specifies the **OpenShiftRoute** resource as the source for the DNS records which gets created in the previously specified DNS provider.

openshiftRouteOptions.routerName
: If the source is **OpenShiftRoute**, then you can pass the OpenShift Ingress Controller name. External DNS Operator selects the canonical hostname of that router as the target while creating the CNAME record.

6. Check the records created for OpenShift Container Platform routes by using the following command:

```
$ aws route53 list-resource-record-sets --hosted-zone-id Z02355203TNN1XXXX1J6O --
query "ResourceRecordSets[?Type == 'CNAME']" | grep console
```

4.5.2. Creating DNS records in a different AWS account by using a shared VPC

To create DNS records in a different AWS account, configure the ExternalDNS Operator to use a shared Virtual Private Cloud (VPC). Your organization can then use a single Route 53 instance for name resolution across multiple accounts and projects.

Prerequisites

- You have created two Amazon AWS accounts: one with a VPC and a Route 53 private hosted zone configured (Account A), and another for installing a cluster (Account B).
- You have created an IAM Policy and IAM Role with the appropriate permissions in Account A for Account B to create DNS records in the Route 53 hosted zone of Account A.
- You have installed a cluster in Account B into the existing VPC for Account A.
- You have installed the ExternalDNS Operator in the cluster in Account B.

Procedure

1. Get the Role ARN of the IAM Role that you created to allow Account B to access Account A's Route 53 hosted zone by running the following command:

```
$ aws --profile account-a iam get-role --role-name user-rol1 | head -1
```

Example output

```
ROLE arn:aws:iam::1234567890123:role/user-rol1 2023-09-14T17:21:54+00:00 3600 /
AROA3SGB2ZRKRT5NISNJN user-rol1
```

2. Locate the private hosted zone to use with Account A's credentials by running the following command:

```
$ aws --profile account-a route53 list-hosted-zones | grep
testextdnsoperator.apacshift.support
```

Example output

```
HOSTEDZONES terraform /hostedzone/Z02355203TNN1XXXX1J6O
testextdnsoperator.apacshift.support. 5
```

3. Create the **ExternalDNS** object by running the following command:

```
$ cat <<EOF | oc create -f -
apiVersion: externaldns.olm.openshift.io/v1beta1
kind: ExternalDNS
metadata:
```

```
  name: sample-aws
spec:
  domains:
  - filterType: Include
    matchType: Exact
    name: testextdnsoperator.apacshift.support
  provider:
    type: AWS
    aws:
      assumeRole:
        arn: arn:aws:iam::12345678901234:role/user-rol1
  source:
    type: OpenShiftRoute
    openshiftRouteOptions:
      routerName: default
EOF
```

where:

arn

Specifies the Role ARN to have DNS records created in Account A.

4. Check the records created for OpenShift Container Platform routes by entering the following command:

```
$ aws --profile account-a route53 list-resource-record-sets --hosted-zone-id
Z02355203TNN1XXXX1J6O --query "ResourceRecordSets[?Type == 'CNAME']" | grep
console-openshift-console
```

4.6. CREATING DNS RECORDS ON AZURE

To create DNS records on Microsoft Azure, use the External DNS Operator. By using this Operator, you can manage external name resolution for your cluster services.

> **IMPORTANT**
>
> Using the External DNS Operator on a Microsoft Entra Workload ID-enabled cluster or a cluster that runs in Microsoft Azure Government (MAG) regions is not supported.

4.6.1. Creating DNS records on an Azure DNS zone

To create DNS records on a public or private DNS zone for Azure, use the External DNS Operator. The Operator manages external name resolution for your cluster.

Prerequisites

- You must have administrator privileges.
- The **admin** user must have access to the **kube-system** namespace.

Procedure

1. Fetch the credentials from the **kube-system** namespace to use the cloud provider client by running the following command:

```
$ CLIENT_ID=$(oc get secrets azure-credentials  -n kube-system  --template=
{{.data.azure_client_id}} | base64 -d)
```

```
$ CLIENT_SECRET=$(oc get secrets azure-credentials  -n kube-system  --template=
{{.data.azure_client_secret}} | base64 -d)
```

```
$ RESOURCE_GROUP=$(oc get secrets azure-credentials  -n kube-system  --template=
{{.data.azure_resourcegroup}} | base64 -d)
```

```
$ SUBSCRIPTION_ID=$(oc get secrets azure-credentials  -n kube-system  --template=
{{.data.azure_subscription_id}} | base64 -d)
```

```
$ TENANT_ID=$(oc get secrets azure-credentials  -n kube-system  --template=
{{.data.azure_tenant_id}} | base64 -d)
```

2. Log in to Azure by running the following command:

```
$ az login --service-principal -u "${CLIENT_ID}" -p "${CLIENT_SECRET}" --tenant
"${TENANT_ID}"
```

3. Get a list of routes by running the following command:

```
$ oc get routes --all-namespaces | grep console
```

Example output

```
openshift-console          console             console-openshift-
console.apps.test.azure.example.com                       console             https   reencrypt/Redirect
None
openshift-console          downloads           downloads-openshift-
console.apps.test.azure.example.com                      downloads           http    edge/Redirect
None
```

4. Get a list of DNS zones.

   a. For public DNS zones, enter the following command:

   ```
   $ az network dns zone list --resource-group "${RESOURCE_GROUP}"
   ```

   b. For private DNS zones, enter the following command:

   ```
   $ az network private-dns zone list -g "${RESOURCE_GROUP}"
   ```

5. Create a YAML file, for example, **external-dns-sample-azure.yaml**, that defines the **ExternalDNS** object:

Example **external-dns-sample-azure.yaml** file

```
apiVersion: externaldns.olm.openshift.io/v1beta1
kind: ExternalDNS
metadata:
```

```
  name: sample-azure
spec:
  zones:
  - "/subscriptions/1234567890/resourceGroups/test-azure-xxxxx-rg/providers/Microsoft.Network/dnszones/test.azure.example.com"
  provider:
    type: Azure
  source:
    openshiftRouteOptions:
      routerName: default
    type: OpenShiftRoute
# ...
```

where:

metadata.name
: Specifies the External DNS name.

spec.zones
: Specifies the zone ID. For a private DNS zone, change **dnszones** to **privateDnsZones**.

provider.type
: Specifies the provider type.

source.openshiftRouteOptions
: Specifies the options for the source of DNS records.

routerName
: If the source type is **OpenShiftRoute**, you can pass the OpenShift Ingress Controller name. The External DNS Operator selects the canonical hostname of that router as the target while creating the CNAME record.

source.type
: Specifies the **route** resource as the source for the Azure DNS records.

Troubleshooting

1. Check the records created for the routes.

   a. For public DNS zones, enter the following command:

   ```
   $ az network dns record-set list -g "${RESOURCE_GROUP}" -z "${ZONE_NAME}" | grep console
   ```

   b. For private DNS zones, enter the following command:

   ```
   $ az network private-dns record-set list -g "${RESOURCE_GROUP}" -z "${ZONE_NAME}" | grep console
   ```

4.7. CREATING DNS RECORDS ON GOOGLE CLOUD PLATFORM

To create DNS records on Google Cloud, use the External DNS Operator. The DNS Operator manages external name resolution for your cluster services.

IMPORTANT

Using the External DNS Operator on a cluster with Google Cloud Workload Identity enabled is not supported. For more information about the Google Cloud Workload Identity, see Google Cloud Workload Identity.

4.7.1. Creating DNS records on a public managed zone for Google Cloud

To create DNS records on Google Cloud, use the External DNS Operator. The DNS Operator manages external name resolution for your cluster services.

Prerequisites

- You must have administrator privileges.

Procedure

1. Copy the **gcp-credentials** secret in the **encoded-gcloud.json** file by running the following command:

```
$ oc get secret gcp-credentials -n kube-system --template='{{$v := index .data
"service_account.json"}}{{$v}}' | base64 -d - > decoded-gcloud.json
```

2. Export your Google credentials by running the following command:

```
$ export GOOGLE_CREDENTIALS=decoded-gcloud.json
```

3. Activate your account by using the following command:

```
$ gcloud auth activate-service-account  <client_email as per decoded-gcloud.json> --key-
file=decoded-gcloud.json
```

4. Set your project by running the following command:

```
$ gcloud config set project <project_id as per decoded-gcloud.json>
```

5. Get a list of routes by running the following command:

```
$ oc get routes --all-namespaces | grep console
```

Example output

```
openshift-console          console             console-openshift-
console.apps.test.gcp.example.com                       console             https   reencrypt/Redirect
None
openshift-console          downloads           downloads-openshift-
console.apps.test.gcp.example.com                       downloads           http    edge/Redirect
None
```

6. Get a list of managed zones, such as **qe-cvs4g-private-zone test.gcp.example.com**, by running the following command:

```
$ gcloud dns managed-zones list | grep test.gcp.example.com
```

■

7. Create a YAML file, for example, **external-dns-sample-gcp.yaml**, that defines the **ExternalDNS** object:

Example **external-dns-sample-gcp.yaml** file

```
apiVersion: externaldns.olm.openshift.io/v1beta1
kind: ExternalDNS
metadata:
  name: sample-gcp
spec:
  domains:
    - filterType: Include
      matchType: Exact
      name: test.gcp.example.com
  provider:
    type: GCP
  source:
    openshiftRouteOptions:
      routerName: default
    type: OpenShiftRoute
# ...
```

where:

metadata.name
: Specifies the External DNS name.

spec.domains.filterType
: By default, all hosted zones are selected as potential targets. You can include your hosted zone.

spec.domains.matchType
: Specifies the domain of the target that must match the string defined by the **name** key.

spec.domains.name
: Specifies the exact domain of the zone you want to update. The hostname of the routes must be subdomains of the specified domain.

spec.provider.type
: Specifies the provider type.

source.openshiftRouteOptions
: Specifies options for the source of DNS records.

openshiftRouteOptions.routerName
: If the source type is **OpenShiftRoute**, you can pass the OpenShift Ingress Controller name. External DNS selects the canonical hostname of that router as the target while creating a CNAME record.

type
: Specifies the **route** resource as the source for Google Cloud DNS records.

8. Check the DNS records created for OpenShift Container Platform routes by running the following command:

```
$ gcloud dns record-sets list --zone=qe-cvs4g-private-zone | grep console
```

4.8. CREATING DNS RECORDS ON INFOBLOX

To create DNS records on Infoblox, use the External DNS Operator. The Operator manages external name resolution for your cluster services.

4.8.1. Creating DNS records on a public DNS zone on Infoblox

To create DNS records on Infoblox, use the External DNS Operator. The Operator manages external name resolution for your cluster services.

Prerequisites

- You have access to the OpenShift CLI (**oc**).
- You have access to the Infoblox UI.

Procedure

1. Create a **secret** object with Infoblox credentials by running the following command:

```
$ oc -n external-dns-operator create secret generic infoblox-credentials --from-literal=EXTERNAL_DNS_INFOBLOX_WAPI_USERNAME=<infoblox_username> --from-literal=EXTERNAL_DNS_INFOBLOX_WAPI_PASSWORD=<infoblox_password>
```

2. Get a list of routes by running the following command:

```
$ oc get routes --all-namespaces | grep console
```

Example output

```
openshift-console          console             console-openshift-console.apps.test.example.com
console             https   reencrypt/Redirect     None
openshift-console          downloads           downloads-openshift-console.apps.test.example.com                     downloads           http    edge/Redirect
None
```

3. Create a YAML file, for example, **external-dns-sample-infoblox.yaml**, that defines the **ExternalDNS** object:

Example external-dns-sample-infoblox.yaml file

```
apiVersion: externaldns.olm.openshift.io/v1beta1
kind: ExternalDNS
metadata:
  name: sample-infoblox
spec:
  provider:
    type: Infoblox
    infoblox:
      credentials:
```

```
      name: infoblox-credentials
    gridHost: ${INFOBLOX_GRID_PUBLIC_IP}
    wapiPort: 443
    wapiVersion: "2.3.1"
domains:
- filterType: Include
  matchType: Exact
  name: test.example.com
source:
  type: OpenShiftRoute
  openshiftRouteOptions:
    routerName: default
```

where:

metadata.name
: Specifies the External DNS name.

provider.type
: Specifies the provider type.

source.type
: Specifies options for the source of DNS records.

routerName
: If the source type is **OpenShiftRoute**, you can pass the OpenShift Ingress Controller name. External DNS selects the canonical hostname of that router as the target while creating a CNAME record.

4. Create the **ExternalDNS** resource on Infoblox by running the following command:

```
$ oc create -f external-dns-sample-infoblox.yaml
```

5. From the Infoblox UI, check the DNS records created for **console** routes:
   a. Click **Data Management → DNS → Zones**.
   b. Select the zone name.

4.9. CONFIGURING THE CLUSTER-WIDE PROXY ON THE EXTERNAL DNS OPERATOR

To propagate proxy settings to your deployed Operators, configure the cluster-wide proxy. The Operator Lifecycle Manager (OLM) automatically updates these Operators with the new **HTTP_PROXY**, **HTTPS_PROXY**, and **NO_PROXY** environment variables.

4.9.1. Trusting the certificate authority of the cluster-wide proxy

To enable the External DNS Operator to authenticate with the cluster-wide proxy, configure the Operator to trust the certificate authority (CA) of the proxy. This ensures secure communication when routing DNS traffic through the proxy.

Procedure

1. Create the config map to contain the CA bundle in the **external-dns-operator** namespace by running the following command:

```
$ oc -n external-dns-operator create configmap trusted-ca
```

2. To inject the trusted CA bundle into the config map, add the **config.openshift.io/inject-trusted-cabundle=true** label to the config map by running the following command:

```
$ oc -n external-dns-operator label cm trusted-ca config.openshift.io/inject-trusted-
cabundle=true
```

3. Update the subscription of the External DNS Operator by running the following command:

```
$ oc -n external-dns-operator patch subscription external-dns-operator --type='json' -
p='[{"op": "add", "path": "/spec/config", "value":{"env":
[{"name":"TRUSTED_CA_CONFIGMAP_NAME","value":"trusted-ca"}]}}]'
```

Verification

- After deploying the External DNS Operator, verify that the trusted CA environment variable is added by running the following command. The output must show **trusted-ca** for the **external-dns-operator** deployment.

```
$ oc -n external-dns-operator exec deploy/external-dns-operator -c external-dns-operator --
printenv TRUSTED_CA_CONFIGMAP_NAME
```

CHAPTER 5. METALLB OPERATOR

5.1. ABOUT METALLB AND THE METALLB OPERATOR

In OpenShift Container Platform clusters running on bare metal or without a cloud load balancer, you can use the MetalLB Operator to assign external IP addresses to LoadBalancer services. These services receive external IPs on the host network.

5.1.1. When to use MetalLB

To get fault-tolerant access to applications through an external IP on bare metal in OpenShift Container Platform, you can use MetalLB.

Using MetalLB is valuable when you have a bare-metal cluster, or an infrastructure that is like bare metal, and you want fault-tolerant access to an application through an external IP address.

You must configure your networking infrastructure to ensure that network traffic for the external IP address is routed from clients to the host network for the cluster.

After deploying MetalLB with the MetalLB Operator, when you add a service of type **LoadBalancer**, MetalLB provides a platform-native load balancer.

When external traffic enters your OpenShift Container Platform cluster through a MetalLB **LoadBalancer** service, the return traffic to the client has the external IP address of the load balancer as the source IP.

MetalLB operating in layer2 mode provides support for failover by utilizing a mechanism similar to IP failover. However, instead of relying on the virtual router redundancy protocol (VRRP) and keepalived, MetalLB leverages a gossip-based protocol to identify instances of node failure. When a failover is detected, another node assumes the role of the leader node, and a gratuitous ARP message is dispatched to broadcast this change.

MetalLB operating in layer3 or border gateway protocol (BGP) mode delegates failure detection to the network. The BGP router or routers that the OpenShift Container Platform nodes have established a connection with will identify any node failure and terminate the routes to that node.

Using MetalLB instead of IP failover is preferable for ensuring high availability of pods and services.

5.1.2. MetalLB Operator custom resources

In OpenShift Container Platform, you configure MetalLB deployment and IP advertisement through custom resources that the MetalLB Operator monitors. The resources include **MetalLB**, **IPAddressPool**, **L2Advertisement**, **BGPAdvertisement**, **BGPPeer**, and **BFDProfile**.

MetalLB

When you add a **MetalLB** custom resource to the cluster, the MetalLB Operator deploys MetalLB on the cluster. The Operator only supports a single instance of the custom resource. If the instance is deleted, the Operator removes MetalLB from the cluster.

IPAddressPool

MetalLB requires one or more pools of IP addresses that it can assign to a service when you add a service of type **LoadBalancer**. An **IPAddressPool** includes a list of IP addresses. The list can be a single IP address that is set using a range, such as 1.1.1.1-1.1.1.1, a range specified in CIDR notation, a range specified as a starting and ending address separated by a hyphen, or a combination of the three. An **IPAddressPool** requires a name. The documentation uses names like **doc-example**, **doc-example-reserved**, and **doc-example-ipv6**. The MetalLB **controller** assigns IP addresses from a pool of addresses in an **IPAddressPool**. **L2Advertisement** and **BGPAdvertisement** custom resources enable the advertisement of a given IP from a given pool. You can assign IP addresses from an **IPAddressPool** to services and namespaces by using the **spec.serviceAllocation** specification in the **IPAddressPool** custom resource.

NOTE

A single **IPAddressPool** can be referenced by a L2 advertisement and a BGP advertisement.

BGPPeer

The BGP peer custom resource identifies the BGP router for MetalLB to communicate with, the AS number of the router, the AS number for MetalLB, and customizations for route advertisement. MetalLB advertises the routes for service load-balancer IP addresses to one or more BGP peers.

BFDProfile

The BFD profile custom resource configures Bidirectional Forwarding Detection (BFD) for a BGP peer. BFD provides faster path failure detection than BGP alone provides.

L2Advertisement

The L2Advertisement custom resource advertises an IP coming from an **IPAddressPool** using the L2 protocol.

BGPAdvertisement

The BGPAdvertisement custom resource advertises an IP coming from an **IPAddressPool** using the BGP protocol.

After you add the **MetalLB** custom resource to the cluster and the Operator deploys MetalLB, the **controller** and **speaker** MetalLB software components begin running.

MetalLB validates all relevant custom resources.

5.1.3. MetalLB software components

In OpenShift Container Platform, you get external IPs for LoadBalancer services from two MetalLB components. The controller assigns IPs from address pools, and the speaker advertises them via layer 2 or BGP.

When you install the MetalLB Operator, the **metallb-operator-controller-manager** deployment starts a pod. The pod is the implementation of the Operator. The pod monitors for changes to all the relevant resources.

When the Operator starts an instance of MetalLB, it starts a **controller** deployment and a **speaker** daemon set.

NOTE

You can configure deployment specifications in the MetalLB custom resource to manage how **controller** and **speaker** pods deploy and run in your cluster. For more information about these deployment specifications, see the *Additional resources* section.

controller

The Operator starts the deployment and a single pod. When you add a service of type **LoadBalancer**, Kubernetes uses the **controller** to allocate an IP address from an address pool. In case of a service failure, verify you have the following entry in your **controller** pod logs:

Example output

```
"event":"ipAllocated","ip":"172.22.0.201","msg":"IP address assigned by controller
```

speaker

The Operator starts a daemon set for **speaker** pods. By default, a pod is started on each node in your cluster. You can limit the pods to specific nodes by specifying a node selector in the **MetalLB** custom resource when you start MetalLB. If the **controller** allocated the IP address to the service and service is still unavailable, read the **speaker** pod logs. If the **speaker** pod is unavailable, run the **oc describe pod -n** command.

For layer 2 mode, after the **controller** allocates an IP address for the service, the **speaker** pods use an algorithm to determine which **speaker** pod on which node will announce the load balancer IP address. The algorithm involves hashing the node name and the load balancer IP address. For more information, see "MetalLB and external traffic policy". The **speaker** uses Address Resolution Protocol (ARP) to announce IPv4 addresses and Neighbor Discovery Protocol (NDP) to announce IPv6 addresses.

For Border Gateway Protocol (BGP) mode, after the **controller** allocates an IP address for the service, each **speaker** pod advertises the load balancer IP address with its BGP peers. You can configure which nodes start BGP sessions with BGP peers.

Requests for the load balancer IP address are routed to the node with the **speaker** that announces the IP address. After the node receives the packets, the service proxy routes the packets to an endpoint for the service. The endpoint can be on the same node in the optimal case, or it can be on another node. The service proxy chooses an endpoint each time a connection is established.

5.1.4. MetalLB and external traffic policy

External traffic policy for MetalLB LoadBalancer services determines how the service proxy distributes traffic to pods. Set the policy to **cluster** for uniform distribution or to **local** to preserve client IP addresses.

With layer 2 mode, one node in your cluster receives all the traffic for the service IP address.

With BGP mode, a router on the host network opens a connection to one of the nodes in the cluster for a new client connection.

How your cluster handles the traffic after it enters the node is affected by the external traffic policy.

cluster

This is the default value for **spec.externalTrafficPolicy**.

With the **cluster** traffic policy, after the node receives the traffic, the service proxy distributes the traffic to all the pods in your service. This policy provides uniform traffic distribution across the pods, but it obscures the client IP address and it can appear to the application in your pods that the traffic originates from the node rather than the client.

local

With the **local** traffic policy, after the node receives the traffic, the service proxy only sends traffic to the pods on the same node. For example, if the **speaker** pod on node A announces the external service IP, then all traffic is sent to node A. After the traffic enters node A, the service proxy only sends traffic to pods for the service that are also on node A. Pods for the service that are on additional nodes do not receive any traffic from node A. Pods for the service on additional nodes act as replicas in case failover is needed.

This policy does not affect the client IP address. Application pods can determine the client IP address from the incoming connections.

NOTE

The following information is important when configuring the external traffic policy in BGP mode.

Although MetalLB advertises the load balancer IP address from all the eligible nodes, the number of nodes loadbalancing the service can be limited by the capacity of the router to establish equal-cost multipath (ECMP) routes. If the number of nodes advertising the IP is greater than the ECMP group limit of the router, the router will use less nodes than the ones advertising the IP.

For example, if the external traffic policy is set to **local** and the router has an ECMP group limit set to 16 and the pods implementing a LoadBalancer service are deployed on 30 nodes, this would result in pods deployed on 14 nodes not receiving any traffic. In this situation, it would be preferable to set the external traffic policy for the service to **cluster**.

5.1.5. MetalLB concepts for layer 2 mode

MetalLB in layer 2 mode announces the external IP for a LoadBalancer service from one node via ARP or NDP. All traffic for the service goes through that node, and failover to another node is automatic when the node becomes unavailable.

NOTE

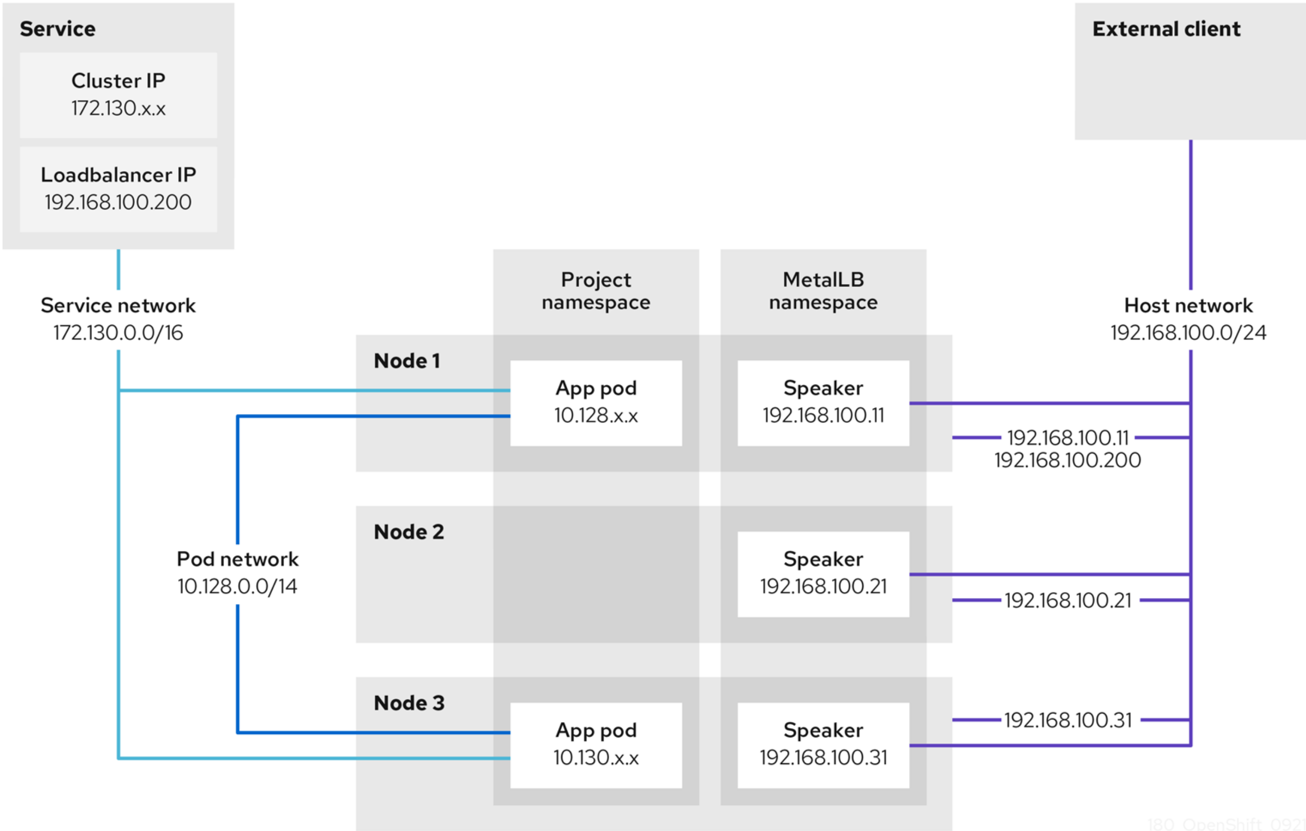
In layer 2 mode, MetalLB relies on ARP and NDP. These protocols implement local address resolution within a specific subnet. In this context, the client must be able to reach the VIP assigned by MetalLB that exists on the same subnet as the nodes announcing the service in order for MetalLB to work.

The **speaker** pod responds to ARP requests for IPv4 services and NDP requests for IPv6.

In layer 2 mode, all traffic for a service IP address is routed through one node. After traffic enters the node, the service proxy for the CNI network provider distributes the traffic to all the pods for the service.

Because all traffic for a service enters through a single node in layer 2 mode, in a strict sense, MetalLB does not implement a load balancer for layer 2. Rather, MetalLB implements a failover mechanism for layer 2 so that when a **speaker** pod becomes unavailable, a **speaker** pod on a different node can announce the service IP address.

When a node becomes unavailable, failover is automatic. The **speaker** pods on the other nodes detect that a node is unavailable and a new **speaker** pod and node take ownership of the service IP address from the failed node.



The preceding graphic shows the following concepts related to MetalLB:

- An application is available through a service that has a cluster IP on the **172.130.0.0/16** subnet. That IP address is accessible from inside the cluster. The service also has an external IP address that MetalLB assigned to the service, **192.168.100.200**.
- Nodes 1 and 3 have a pod for the application.
- The **speaker** daemon set runs a pod on each node. The MetalLB Operator starts these pods.
- Each **speaker** pod is a host-networked pod. The IP address for the pod is identical to the IP address for the node on the host network.
- The **speaker** pod on node 1 uses ARP to announce the external IP address for the service, **192.168.100.200**. The **speaker** pod that announces the external IP address must be on the same node as an endpoint for the service and the endpoint must be in the **Ready** condition.
- Client traffic is routed to the host network and connects to the **192.168.100.200** IP address. After traffic enters the node, the service proxy sends the traffic to the application pod on the same node or another node according to the external traffic policy that you set for the service.
  - If the external traffic policy for the service is set to **cluster**, the node that advertises the **192.168.100.200** load balancer IP address is selected from the nodes where a **speaker** pod is running. Only that node can receive traffic for the service.
  - If the external traffic policy for the service is set to **local**, the node that advertises the

192.168.100.200 load balancer IP address is selected from the nodes where a **speaker** pod is running and at least an endpoint of the service. Only that node can receive traffic for the service. In the preceding graphic, either node 1 or 3 would advertise **192.168.100.200**.

- If node 1 becomes unavailable, the external IP address fails over to another node. On another node that has an instance of the application pod and service endpoint, the **speaker** pod begins to announce the external IP address, **192.168.100.200** and the new node receives the client traffic. In the diagram, the only candidate is node 3.

5.1.6. MetalLB concepts for BGP mode

MetalLB in border gateway protocol (BGP) mode advertises load balancer IP addresses to BGP peers from each **speaker** pod. The router sends traffic to one of the nodes, so load is distributed across nodes and the router switches to another node when one becomes unavailable.

It is also possible to advertise the IPs coming from a given pool to a specific set of peers by adding an optional list of BGP peers.

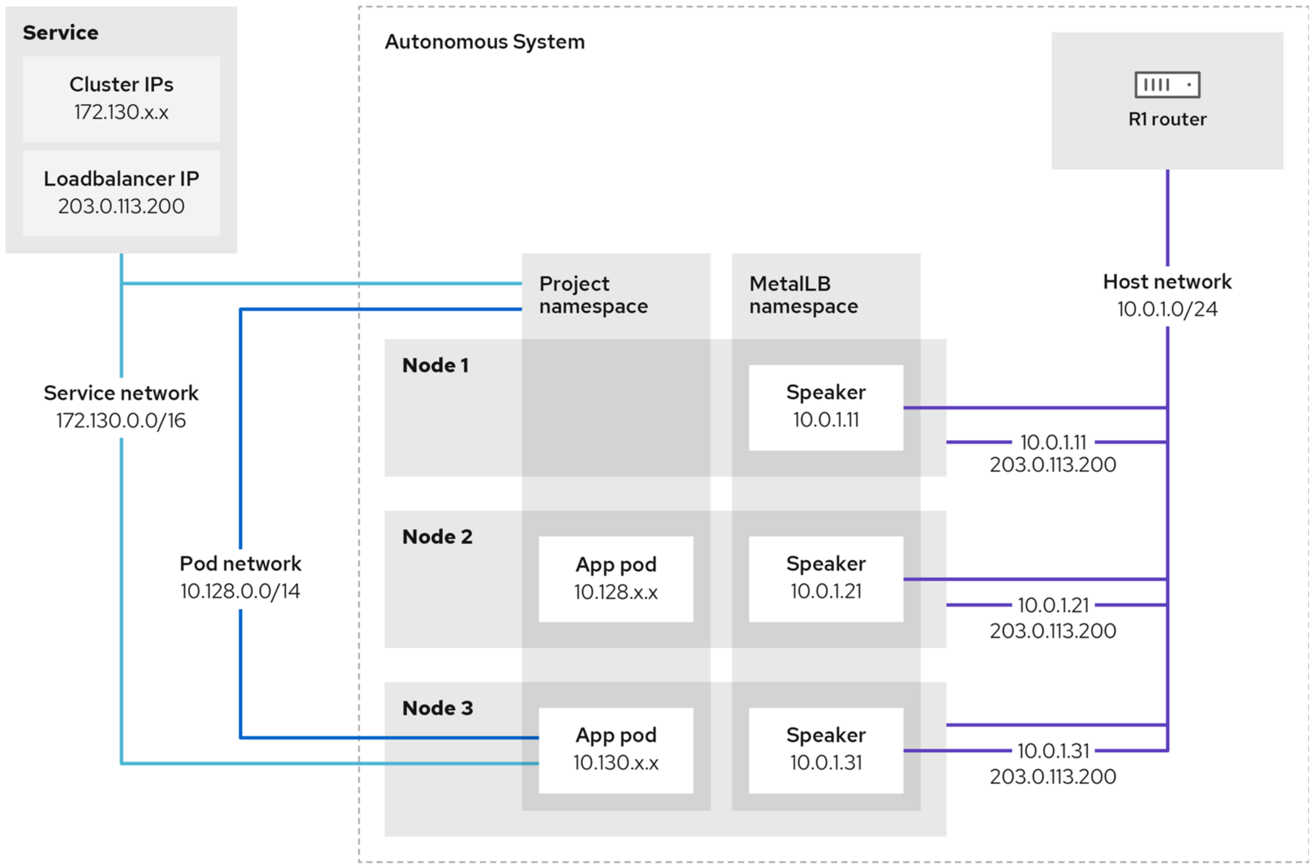
BGP peers are commonly network routers that are configured to use the BGP protocol. When a router receives traffic for the load balancer IP address, the router picks one of the nodes with a **speaker** pod that advertised the IP address. The router sends the traffic to that node. After traffic enters the node, the service proxy for the CNI network plugin distributes the traffic to all the pods for the service.

The directly-connected router on the same layer 2 network segment as the cluster nodes can be configured as a BGP peer. If the directly-connected router is not configured as a BGP peer, you need to configure your network so that packets for load balancer IP addresses are routed between the BGP peers and the cluster nodes that run the **speaker** pods.

Each time a router receives new traffic for the load balancer IP address, it creates a new connection to a node. Each router manufacturer has an implementation-specific algorithm for choosing which node to initiate the connection with. However, the algorithms commonly are designed to distribute traffic across the available nodes for the purpose of balancing the network load.

If a node becomes unavailable, the router initiates a new connection with another node that has a **speaker** pod that advertises the load balancer IP address.

Figure 5.1. MetalLB topology diagram for BGP mode



The preceding graphic shows the following concepts related to MetalLB:

- An application is available through a service that has an IPv4 cluster IP on the **172.130.0.0/16** subnet. That IP address is accessible from inside the cluster. The service also has an external IP address that MetalLB assigned to the service, **203.0.113.200**.
- Nodes 2 and 3 have a pod for the application.
- The **speaker** daemon set runs a pod on each node. The MetalLB Operator starts these pods. You can configure MetalLB to specify which nodes run the **speaker** pods.
- Each **speaker** pod is a host-networked pod. The IP address for the pod is identical to the IP address for the node on the host network.
- Each **speaker** pod starts a BGP session with all BGP peers and advertises the load balancer IP addresses or aggregated routes to the BGP peers. The **speaker** pods advertise that they are part of Autonomous System 65010. The diagram shows a router, R1, as a BGP peer within the same Autonomous System. However, you can configure MetalLB to start BGP sessions with peers that belong to other Autonomous Systems.
- All the nodes with a **speaker** pod that advertises the load balancer IP address can receive traffic for the service.
  - If the external traffic policy for the service is set to **cluster**, all the nodes where a speaker pod is running advertise the **203.0.113.200** load balancer IP address and all the nodes with a **speaker** pod can receive traffic for the service. The host prefix is advertised to the router

peer only if the external traffic policy is set to cluster.

- If the external traffic policy for the service is set to **local**, then all the nodes where a **speaker** pod is running and at least an endpoint of the service is running can advertise the **203.0.113.200** load balancer IP address. Only those nodes can receive traffic for the service. In the preceding graphic, nodes 2 and 3 would advertise **203.0.113.200**.

- You can configure MetalLB to control which **speaker** pods start BGP sessions with specific BGP peers by specifying a node selector when you add a BGP peer custom resource.
- Any routers, such as R1, that are configured to use BGP can be set as BGP peers.
- Client traffic is routed to one of the nodes on the host network. After traffic enters the node, the service proxy sends the traffic to the application pod on the same node or another node according to the external traffic policy that you set for the service.
- If a node becomes unavailable, the router detects the failure and initiates a new connection with another node. You can configure MetalLB to use a Bidirectional Forwarding Detection (BFD) profile for BGP peers. BFD provides faster link failure detection so that routers can initiate new connections earlier than without BFD.

5.1.7. Limitations and restrictions

MetalLB has limitations for infrastructure, layer 2 mode, and BGP mode in OpenShift Container Platform. Consider infrastructure fit, layer 2 single-node bandwidth and failover, and BGP connection resets and single ASN when you plan your deployment.

5.1.7.1. Infrastructure considerations for MetalLB

You can use MetalLB for bare metal and on-premise installations that lack a native load balancer.

In addition to bare-metal installations, installations of OpenShift Container Platform on some infrastructures might not include a native load-balancer capability. For example, the following infrastructures can benefit from adding the MetalLB Operator:

- Bare metal
- VMware vSphere
- IBM Z® and IBM® LinuxONE
- IBM Z® and IBM® LinuxONE for Red Hat Enterprise Linux (RHEL) KVM
- IBM Power®

5.1.7.2. Limitations for layer 2 mode

In OpenShift Container Platform, MetalLB layer 2 mode is limited to single-node bandwidth and failover depends on client ARP handling. Avoid using the same VLAN for MetalLB and an additional network to prevent connection failures.

5.1.7.2.1. Single-node bottleneck

MetalLB routes all traffic for a service through a single node, the node can become a bottleneck and limit performance.

Layer 2 mode limits the ingress bandwidth for your service to the bandwidth of a single node. This is a fundamental limitation of using ARP and NDP to direct traffic.

5.1.7.2.2. Slow failover performance

Failover between nodes depends on cooperation from the clients. When a failover occurs, MetalLB sends gratuitous ARP packets to notify clients that the MAC address associated with the service IP has changed.

Most client operating systems handle gratuitous ARP packets correctly and update their neighbor caches promptly. When clients update their caches quickly, failover completes within a few seconds. Clients typically fail over to a new node within 10 seconds. However, some client operating systems either do not handle gratuitous ARP packets at all or have outdated implementations that delay the cache update.

Recent versions of common operating systems such as Windows, macOS, and Linux implement layer 2 failover correctly. Issues with slow failover are not expected except for older and less common client operating systems.

To minimize the impact from a planned failover on outdated clients, keep the old node running for a few minutes after flipping leadership. The old node can continue to forward traffic for outdated clients until their caches refresh.

During an unplanned failover, the service IPs are unreachable until the outdated clients refresh their cache entries.

5.1.7.2.3. Additional Network and MetalLB cannot use same network

Using the same VLAN for both MetalLB and an additional network interface set up on a source pod might result in a connection failure. This occurs when both the MetalLB IP and the source pod reside on the same node.

To avoid connection failures, place the MetalLB IP in a different subnet from the one where the source pod resides. This configuration ensures that traffic from the source pod will take the default gateway. Consequently, the traffic can effectively reach its destination by using the OVN overlay network, ensuring that the connection functions as intended.

5.1.7.3. Limitations for BGP mode

In OpenShift Container Platform, MetalLB border gateway protocol (BGP) mode can reset active connections when a BGP session terminates and requires a single ASN and router ID for all BGP peers. Use a node selector when adding a BGP peer to limit which nodes run BGP sessions and reduce the impact of node faults.

5.1.7.3.1. Node failure can break all active connections

MetalLB shares a limitation that is common to BGP-based load balancing. When a BGP session terminates, such as when a node fails or when a **speaker** pod restarts, the session termination might result in resetting all active connections. End users can experience a **Connection reset by peer** message.

The consequence of a terminated BGP session is implementation-specific for each router manufacturer. However, you can anticipate that a change in the number of **speaker** pods affects the number of BGP sessions and that active connections with BGP peers will break.

To avoid or reduce the likelihood of a service interruption, you can specify a node selector when you add a BGP peer. By limiting the number of nodes that start BGP sessions, a fault on a node that does not have a BGP session has no affect on connections to the service.

5.1.7.3.2. Support for a single ASN and a single router ID only

When you add a BGP peer custom resource, you specify the **spec.myASN** field to identify the Autonomous System Number (ASN) that MetalLB belongs to. OpenShift Container Platform uses an implementation of BGP with MetalLB that requires MetalLB to belong to a single ASN. If you attempt to add a BGP peer and specify a different value for **spec.myASN** than an existing BGP peer custom resource, you receive an error.

Similarly, when you add a BGP peer custom resource, the **spec.routerID** field is optional. If you specify a value for this field, you must specify the same value for all other BGP peer custom resources that you add.

The limitation to support a single ASN and single router ID is a difference with the community-supported implementation of MetalLB.

5.1.8. Additional resources

- Comparison: Fault tolerant access to external IP addresses
- Removing IP failover
- Deployment specifications for MetalLB

5.2. INSTALLING THE METALLB OPERATOR

As a cluster administrator, you can add the MetalLB Operator so that the Operator can manage the lifecycle for an instance of MetalLB on your cluster.

MetalLB and IP failover are incompatible. If you configured IP failover for your cluster, perform the steps to remove IP failover before you install the Operator.

5.2.1. Installing the MetalLB Operator from the OperatorHub by using the web console

As a cluster administrator, you can install the MetalLB Operator by using the OpenShift Container Platform web console.

Prerequisites

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

Procedure

1. In the OpenShift Container Platform web console, navigate to **Operators → OperatorHub**.
2. Type a keyword into the **Filter by keyword** box or scroll to find the Operator you want. For example, type **metallb** to find the MetalLB Operator.
   You can also filter options by **Infrastructure Features**. For example, select **Disconnected** if you want to see Operators that work in disconnected environments, also known as restricted network environments.

3. On the **Install Operator** page, accept the defaults and click **Install**.

Verification

1. To confirm that the installation is successful:

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

    b. Check that the Operator is installed in the **openshift-operators** namespace and that its status is **Succeeded**.

2. If the Operator is not installed successfully, check the status of the Operator and review the logs:

    a. Navigate to the **Operators** → **Installed Operators** page and inspect the **Status** column for any errors or failures.

    b. Navigate to the **Workloads** → **Pods** page and check the logs in any pods in the **openshift-operators** project that are reporting issues.

5.2.2. Installing from OperatorHub using the CLI

To install the MetalLB Operator from the software catalog in OpenShift Container Platform without using the web console, you can use the OpenShift CLI (**oc**).

It is recommended that when using the CLI you install the Operator in the **metallb-system** namespace.

Prerequisites

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

Procedure

1. Create a namespace for the MetalLB Operator by entering the following command:

```
$ cat << EOF | oc apply -f -
apiVersion: v1
kind: Namespace
metadata:
  name: metallb-system
EOF
```

2. Create an Operator group custom resource (CR) in the namespace:

```
$ cat << EOF | oc apply -f -
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: metallb-operator
  namespace: metallb-system
EOF
```

–

3. Confirm the Operator group is installed in the namespace:

```
$ oc get operatorgroup -n metallb-system
```

Example output

```
NAME               AGE
metallb-operator   14m
```

4. Create a **Subscription** CR:

   a. Define the **Subscription** CR and save the YAML file, for example, **metallb-sub.yaml**:

   ```
   apiVersion: operators.coreos.com/v1alpha1
   kind: Subscription
   metadata:
     name: metallb-operator-sub
     namespace: metallb-system
   spec:
     channel: stable
     name: metallb-operator
     source: redhat-operators
     sourceNamespace: openshift-marketplace
   ```

   - For the **spec.source** parameter, must specify the **redhat-operators** value.

   b. To create the **Subscription** CR, run the following command:

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

5. Optional: To ensure BGP and BFD metrics appear in Prometheus, you can label the namespace as in the following command:

```
$ oc label ns metallb-system "openshift.io/cluster-monitoring=true"
```

Verification

The verification steps assume the MetalLB Operator is installed in the **metallb-system** namespace.

1. Confirm the install plan is in the namespace:

```
$ oc get installplan -n metallb-system
```

Example output

```
NAME            CSV                                      APPROVAL    APPROVED
install-wzg94   metallb-operator.4.18.0-nnnnnnnnnnnn   Automatic   true
```

> **NOTE**
>
> Installation of the Operator might take a few seconds.

2. To verify that the Operator is installed, enter the following command and then check that output shows **Succeeded** for the Operator:

```
$ oc get clusterserviceversion -n metallb-system \
  -o custom-columns=Name:.metadata.name,Phase:.status.phase
```

5.2.3. Starting MetalLB on your cluster

To start MetalLB on your cluster after installing the MetalLB Operator in OpenShift Container Platform, you create a single MetalLB custom resource.

Prerequisites

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

Procedure

1. Create a single instance of a MetalLB custom resource:

```
$ cat << EOF | oc apply -f -
apiVersion: metallb.io/v1beta1
kind: MetalLB
metadata:
  name: metallb
  namespace: metallb-system
EOF
```

- For the **metdata.namespace** parameter, substitute **metallb-system** with **openshift-operators** if you installed the MetalLB Operator using the web console.

Verification

Confirm that the deployment for the MetalLB controller and the daemon set for the MetalLB speaker are running.

1. Verify that the deployment for the controller is running:

```
$ oc get deployment -n metallb-system controller
```

Example output

```
NAME        READY   UP-TO-DATE   AVAILABLE   AGE
controller   1/1     1            1           11m
```

2. Verify that the daemon set for the speaker is running:

```
$ oc get daemonset -n metallb-system speaker
```

Example output

```
NAME    DESIRED  CURRENT  READY  UP-TO-DATE  AVAILABLE  NODE
SELECTOR         AGE
speaker  6        6        6      6           6          kubernetes.io/os=linux  18m
```

The example output indicates 6 speaker pods. The number of speaker pods in your cluster might differ from the example output. Make sure the output indicates one pod for each node in your cluster.

5.2.4. Deployment specifications for MetalLB

Deployment specifications in the **MetalLB** custom resource control how the MetalLB **controller** and **speaker** pods deploy and run in OpenShift Container Platform.

Use deployment specifications to manage the following tasks:

- Select nodes for MetalLB pod deployment.
- Manage scheduling by using pod priority and pod affinity.
- Assign CPU limits for MetalLB pods.
- Assign a container RuntimeClass for MetalLB pods.
- Assign metadata for MetalLB pods.

5.2.4.1. Limit speaker pods to specific nodes

You can limit MetalLB **speaker** pods to specific nodes in OpenShift Container Platform by configuring a node selector in the **MetalLB** custom resource. Only nodes that run a **speaker** pod advertise load balancer IP addresses, so you control which nodes serve MetalLB traffic.

The most common reason to limit the **speaker** pods to specific nodes is to ensure that only nodes with network interfaces on specific networks advertise load balancer IP addresses.

If you limit the **speaker** pods to specific nodes and specify **local** for the external traffic policy of a service, then you must ensure that the application pods for the service are deployed to the same nodes.

Example configuration to limit speaker pods to worker nodes

```yaml
apiVersion: metallb.io/v1beta1
kind: MetalLB
metadata:
  name: metallb
  namespace: metallb-system
spec:
  nodeSelector:
    node-role.kubernetes.io/worker: ""
  speakerTolerations:
  - key: "Example"
    operator: "Exists"
    effect: "NoExecute"
```

- In this example configuration, the **spec.nodeSelector** field assigns the **speaker** pods to worker nodes. You can specify labels that you assigned to nodes or any valid node selector.

- In this example configuration, **spec.speakerToTolerations** pod that this toleration is attached to tolerates any taint that matches the **key** and **effect** values by using the **operator** value.

After you apply a manifest with the **spec.nodeSelector** field, you can check the number of pods that the Operator deployed with the **oc get daemonset -n metallb-system speaker** command. Similarly, you can display the nodes that match your labels with a command like **oc get nodes -l node-role.kubernetes.io/worker=**.

You can optionally allow the node to control which speaker pods should, or should not, be scheduled on them by using affinity rules. You can also limit these pods by applying a list of tolerations. For more information about affinity rules, taints, and tolerations, see the additional resources.

5.2.4.2. Configuring pod priority and pod affinity in a MetalLB deployment

To control scheduling of MetalLB controller and **speaker** pods in OpenShift Container Platform, you can assign pod priority and pod affinity in the **MetalLB** custom resource. You create a **PriorityClass** and set **priorityClassName** and affinity in the **MetalLB** spec, then apply the configuration.

The pod priority indicates the relative importance of a pod on a node and schedules the pod based on this priority. Set a high priority on your **controller** or **speaker** pod to ensure scheduling priority over other pods on the node.

Pod affinity manages relationships among pods. Assign pod affinity to the **controller** or **speaker** pods to control on what node the scheduler places the pod in the context of pod relationships. For example, you can use pod affinity rules to ensure that certain pods are located on the same node or nodes, which can help improve network communication and reduce latency between those components.

Prerequisites

- You are logged in as a user with **cluster-admin** privileges.
- You have installed the MetalLB Operator.
- You have started the MetalLB Operator on your cluster.

Procedure

1. Create a **PriorityClass** custom resource, such as **myPriorityClass.yaml**, to configure the priority level. This example defines a **PriorityClass** named **high-priority** with a value of **1000000**. Pods that are assigned this priority class are considered higher priority during scheduling compared to pods with lower priority classes:

```
apiVersion: scheduling.k8s.io/v1
kind: PriorityClass
metadata:
  name: high-priority
value: 1000000
```

2. Apply the **PriorityClass** custom resource configuration:

```
$ oc apply -f myPriorityClass.yaml
```

3. Create a **MetalLB** custom resource, such as **MetalLBPodConfig.yaml**, to specify the **priorityClassName** and **podAffinity** values:

```
apiVersion: metallb.io/v1beta1
kind: MetalLB
metadata:
  name: metallb
  namespace: metallb-system
spec:
  logLevel: debug
  controllerConfig:
    priorityClassName: high-priority
    affinity:
      podAffinity:
        requiredDuringSchedulingIgnoredDuringExecution:
        - labelSelector:
            matchLabels:
             app: metallb
          topologyKey: kubernetes.io/hostname
  speakerConfig:
    priorityClassName: high-priority
    affinity:
      podAffinity:
        requiredDuringSchedulingIgnoredDuringExecution:
        - labelSelector:
            matchLabels:
             app: metallb
          topologyKey: kubernetes.io/hostname
```

where:

spec.controllerConfig.priorityClassName

Specifies the priority class for the MetalLB controller pods. In this case, it is set to **high-priority**.

spec.controllerConfig.affinity.podAffinity

Specifies that you are configuring pod affinity rules. These rules dictate how pods are scheduled in relation to other pods or nodes. This configuration instructs the scheduler to schedule pods that have the label **app: metallb** onto nodes that share the same hostname. This helps to co-locate MetalLB-related pods on the same nodes, potentially optimizing network communication, latency, and resource usage between these pods.

4. Apply the **MetalLB** custom resource configuration by running the following command:

```
$ oc apply -f MetalLBPodConfig.yaml
```

Verification

- To view the priority class that you assigned to pods in the **metallb-system** namespace, run the following command:

```
$ oc get pods -n metallb-system -o custom-
columns=NAME:.metadata.name,PRIORITY:.spec.priorityClassName
```

Example output

```
NAME                                                    PRIORITY
```

```
controller-584f5c8cd8-5zbvg                       high-priority
metallb-operator-controller-manager-9c8d9985-szkqg   <none>
metallb-operator-webhook-server-c895594d4-shjgx      <none>
speaker-dddf7                                     high-priority
```

- Verify that the scheduler placed pods according to pod affinity rules by viewing the metadata for the node of the pod. For example:

```
$ oc get pod -o=custom-columns=NODE:.spec.nodeName,NAME:.metadata.name -n
metallb-system
```

5.2.4.3. Configuring pod CPU limits in a MetalLB deployment

To manage compute resources on nodes running MetalLB in OpenShift Container Platform, you can assign CPU limits to the **controller** and **speaker** pods in the **MetalLB** custom resource. This ensures that all pods on the node have the necessary compute resources to manage workloads and cluster housekeeping.

Prerequisites

- You are logged in as a user with **cluster-admin** privileges.
- You have installed the MetalLB Operator.

Procedure

1. Create a **MetalLB** custom resource file, such as **CPULimits.yaml**, to specify the **cpu** value for the **controller** and **speaker** pods:

```yaml
apiVersion: metallb.io/v1beta1
kind: MetalLB
metadata:
  name: metallb
  namespace: metallb-system
spec:
  logLevel: debug
  controllerConfig:
    resources:
      limits:
        cpu: "200m"
  speakerConfig:
    resources:
      limits:
        cpu: "300m"
```

2. Apply the **MetalLB** custom resource configuration:

```
$ oc apply -f CPULimits.yaml
```

Verification

- To view compute resources for a pod, run the following command, replacing **<pod_name>** with your target pod:

```
$ oc describe pod <pod_name>
```

5.2.5. Additional resources

- Placing pods on specific nodes using node selectors
- Controlling pod placement using node taints
- Understanding pod priority
- Understanding pod affinity

5.3. UPGRADING THE METALLB OPERATOR

The **Subscription** custom resource (CR) for the MetalLB Operator is used to manage whether the Operator is upgraded automatically or manually.

By default, the **Subscription** CR assigns the namespace to **metallb-system** and automatically sets the **installPlanApproval** parameter to **Automatic**. This means that when Red Hat-provided Operator catalogs include a newer version of the MetalLB Operator, the MetalLB Operator is automatically upgraded.

If you need to manually control upgrading the MetalLB Operator, set the **installPlanApproval** parameter to **Manual**.

5.3.1. Manually upgrading the MetalLB Operator

To manually control when the MetalLB Operator upgrades in OpenShift Container Platform, you set **installPlanApproval** to Manual in the Subscription custom resource and approve the install plan. You then verify the upgrade by using the **ClusterServiceVersion** status.

Prerequisites

- You updated your cluster to the latest z-stream release.
- You used OperatorHub to install the MetalLB Operator.
- Access the cluster as a user with the **cluster-admin** role.

Procedure

1. Get the YAML definition of the **metallb-operator** subscription in the **metallb-system** namespace by entering the following command:

   ```
   $ oc -n metallb-system get subscription metallb-operator -o yaml
   ```

2. Edit the **Subscription** CR by setting the **installPlanApproval** parameter to **Manual**:

   ```
   apiVersion: operators.coreos.com/v1alpha1
   kind: Subscription
   metadata:
     name: metallb-operator
     namespace: metallb-system
   ```

```
# ...
spec:
  channel: stable
  installPlanApproval: Manual
  name: metallb-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
# ...
```

3. Find the latest OpenShift Container Platform 4.18 version of the MetalLB Operator by entering the following command:

```
$ oc -n metallb-system get csv
```

4. Check the install plan that exists in the namespace by entering the following command.

```
$ oc -n metallb-system get installplan
```

Example output that shows install-tsz2g as a manual install plan

```
NAME            CSV                                     APPROVAL    APPROVED
install-shpmd   metallb-operator.v4.18.0-202502261233   Automatic   true
install-tsz2g   metallb-operator.v4.18.0-202503102139   Manual      false
```

5. Edit the install plan that exists in the namespace by entering the following command. Ensure that you replace **<name_of_installplan>** with the name of the install plan, such as **install-tsz2g**.

```
$ oc edit installplan <name_of_installplan> -n metallb-system
```

   a. With the install plan open in your editor, set the **spec.approval** parameter to **Manual** and set the **spec.approved** parameter to **true**.

> **NOTE**
>
> After you edit the install plan, the upgrade operation starts. If you enter the **oc -n metallb-system get csv** command during the upgrade operation, the output might show the **Replacing** or the **Pending** status.

Verification

- To verify that the Operator is upgraded, enter the following command and then check that output shows **Succeeded** for the Operator:

```
$ oc -n metallb-system get csv
```

5.3.2. Additional resources

- Introduction to OpenShift updates
- Installing the MetalLB Operator

CHAPTER 6. CLUSTER NETWORK OPERATOR IN OPENSHIFT CONTAINER PLATFORM

With the Cluster Network Operator, you can manage networking in OpenShift Container Platform, including how to view status, enable IP forwarding, and collect logs.

You can use the Cluster Network Operator (CNO) to deploy and manage cluster network components on an OpenShift Container Platform cluster, including the Container Network Interface (CNI) network plugin selected for the cluster during installation.

6.1. CLUSTER NETWORK OPERATOR

The Cluster Network Operator implements the **network** API from the **operator.openshift.io** API group. The Operator deploys the OVN-Kubernetes network plugin, or the network provider plugin that you selected during cluster installation, by using a daemon set.

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

Procedure

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   4.16.1    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.

6.2. VIEWING THE CLUSTER NETWORK CONFIGURATION

You can view your OpenShift Container Platform cluster network configuration by using the **oc describe** command for the **network.config/cluster** resource.

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:
  Creation Timestamp:  2024-08-08T11:25:56Z
  Generation:          3
  Resource Version:    29821
  UID:                 808dd2be-5077-4ff7-b6bb-21b7110126c7
Spec:
  Cluster Network:
    Cidr:         10.128.0.0/14
    Host Prefix:  23
  External IP:
    Policy:
  Network Diagnostics:
    Mode:
    Source Placement:
    Target Placement:
  Network Type:  OVNKubernetes
  Service Network:
    172.30.0.0/16
Status
  Cluster Network:
    Cidr:               10.128.0.0/14
    Host Prefix:        23
  Cluster Network MTU:  1360
  Conditions:
    Last Transition Time:  2024-08-08T11:51:50Z
    Message:
    Observed Generation:   0
    Reason:                AsExpected
    Status:                True
    Type:                  NetworkDiagnosticsAvailable
  Network Type:            OVNKubernetes
  Service Network:
    172.30.0.0/16
Events:  <none>
```

where:

spec

Specifies the field that displays the configured state of the cluster network.

Status

Displays the current state of the cluster network configuration.

6.3. VIEWING CLUSTER NETWORK OPERATOR STATUS

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

Procedure

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

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

6.4. ENABLING IP FORWARDING GLOBALLY

From OpenShift Container Platform 4.14 onward, OVN-Kubernetes disables global IP forwarding by default. By setting the Cluster Network Operator **gatewayConfig.ipForwarding** spec to **Global**, you can enable cluster-wide forwarding.

Procedure

1. Backup the existing network configuration by running the following command:

```
$ oc get network.operator cluster -o yaml > network-config-backup.yaml
```

2. Run the following command to modify the existing network configuration:

```
$ oc edit network.operator cluster
```

a. Add or update the following block under **spec** as illustrated in the following example:

```
spec:
  clusterNetwork:
  - cidr: 10.128.0.0/14
    hostPrefix: 23
  serviceNetwork:
  - 172.30.0.0/16
  networkType: OVNKubernetes
  clusterNetworkMTU: 8900
  defaultNetwork:
    ovnKubernetesConfig:
      gatewayConfig:
        ipForwarding: Global
```

b. Save and close the file.

3. After applying the changes, the OpenShift Cluster Network Operator (CNO) applies the update across the cluster. You can monitor the progress by using the following command:

```
$ oc get clusteroperators network
```

The status should eventually report as **Available**, **Progressing=False**, and **Degraded=False**.

4. Alternatively, you can enable IP forwarding globally by running the following command:

```
$ oc patch network.operator cluster -p '{"spec":{"defaultNetwork":{"ovnKubernetesConfig":
{"gatewayConfig":{"ipForwarding": "Global"}}}}}' --type=merge
```

> **NOTE**
>
> The other valid option for this parameter is **Restricted** in case you want to revert this change. **Restricted** is the default and with that setting global IP address forwarding is disabled.

6.5. 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
```

6.6. CLUSTER NETWORK OPERATOR CONFIGURATION

To manage cluster networking, configure the Cluster Network Operator (CNO) **Network** custom resource (CR) named **cluster** so the cluster uses the correct IP ranges and network plugin settings for reliable pod and service connectivity. Some settings and fields are inherited at the time of install or by the **default.Network.type** plugin, OVN-Kubernetes.

The CNO configuration inherits the following fields during cluster installation from the **Network** API in the **Network.config.openshift.io** API group:

clusterNetwork
: IP address pools from which pod IP addresses are allocated.

serviceNetwork
: IP address pool for services.

defaultNetwork.type
: Cluster network plugin. **OVNKubernetes** is the only supported plugin during installation.

> **NOTE**
>
> After cluster installation, you can only modify the **clusterNetwork** IP address range.

You can specify the cluster network plugin configuration for your cluster by setting the fields for the **defaultNetwork** object in the CNO object named **cluster**.

6.6.1. Cluster Network Operator configuration object

The fields for the Cluster Network Operator (CNO) are described in the following table:

Table 6.1. Cluster Network Operator configuration object

Field	Type	Description
metadata.name	**string**	The name of the CNO object. This name is always **cluster**.

Field	Type	Description

spec.clusterNetwork	**array**	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 in the cluster. For example:
		`spec:`
		`  clusterNetwork:`
		`  - cidr: 10.128.0.0/19`
		`    hostPrefix: 23`
		`  - cidr: 10.128.32.0/19`
		`    hostPrefix: 23`

spec.serviceNetwork	**array**	A block of IP addresses for services. The OVN-Kubernetes network plugin supports only a single IP address block for the service network. For example:
		`spec:`
		`  serviceNetwork:`
		`  - 172.30.0.0/14`
		This value is ready-only and inherited from the **Network.config.openshift.io** object named **cluster** during cluster installation.

spec.defaultNetwork	**object**	Configures the network plugin for the cluster network.

IMPORTANT

For a cluster that needs to deploy objects across multiple networks, ensure that you specify the same value for the **clusterNetwork.hostPrefix** parameter for each network type that is defined in the **install-config.yaml** file. Setting a different value for each **clusterNetwork.hostPrefix** parameter can impact the OVN-Kubernetes network plugin, where the plugin cannot effectively route object traffic among different nodes.

6.6.2. defaultNetwork object configuration

The values for the **defaultNetwork** object are defined in the following table:

Table 6.2. defaultNetwork object

Field	Type	Description

Field	Type	Description
type	**string**	**OVNKubernetes**. The Red Hat OpenShift Networking network plugin is selected during installation. This value cannot be changed after cluster installation. NOTE OpenShift Container Platform uses the OVN-Kubernetes network plugin by default.
ovnKubernetesConfig	**object**	This object is only valid for the OVN-Kubernetes network plugin.

6.6.3. Configuration for the OVN-Kubernetes network plugin

The following table describes the configuration fields for the OVN-Kubernetes network plugin:

Table 6.3. **ovnKubernetesConfig** object

Field	Type	Description
mtu	**integer**	The maximum transmission unit (MTU) for the Geneve (Generic Network Virtualization Encapsulation) overlay network. This value is normally configured automatically.
genevePort	**integer**	The UDP port for the Geneve overlay network.
ipsecConfig	**object**	An object describing the IPsec mode for the cluster.
ipv4	**object**	Specifies a configuration object for IPv4 settings.
ipv6	**object**	Specifies a configuration object for IPv6 settings.
policyAuditConfig	**object**	Specify a configuration object for customizing network policy audit logging. If unset, the defaults audit log settings are used.

Field	Type	Description
gatewayConfig	**object**	Optional: Specify a configuration object for customizing how egress traffic is sent to the node gateway. Valid values are **Shared** and **Local**. The default value is **Shared**. In the default setting, the Open vSwitch (OVS) outputs traffic directly to the node IP interface. In the **Local** setting, it traverses the host network; consequently, it gets applied to the routing table of the host. NOTE While migrating egress traffic, you can expect some disruption to workloads and service traffic until the Cluster Network Operator (CNO) successfully rolls out the changes.

Table 6.4. **ovnKubernetesConfig.ipv4** object

Field	Type	Description
internalTransitSwitchSubnet	string	If your existing network infrastructure overlaps with the **100.88.0.0/16** IPv4 subnet, you can specify a different IP address range for internal use by OVN-Kubernetes. The subnet for the distributed transit switch that enables east-west traffic. This subnet cannot overlap with any other subnets used by OVN-Kubernetes or on the host itself. It must be large enough to accommodate one IP address per node in your cluster. The default value is **100.88.0.0/16**.
internalJoinSubnet	string	If your existing network infrastructure overlaps with the **100.64.0.0/16** IPv4 subnet, you can specify a different IP address range for internal use by OVN-Kubernetes. You must ensure that the IP address range does not overlap with any other subnet used by your OpenShift Container Platform installation. The IP address range must be larger than the maximum number of nodes that can be added to the cluster. For example, if the **clusterNetwork.cidr** value is **10.128.0.0/14** and the **clusterNetwork.hostPrefix** value is **/23**, then the maximum number of nodes is **2^(23-14)=512**. The default value is **100.64.0.0/16**.

Table 6.5. **ovnKubernetesConfig.ipv6** object

Field	Type	Description
internalTransitSwitchSubnet	string	If your existing network infrastructure overlaps with the **fd97::/64** IPv6 subnet, you can specify a different IP address range for internal use by OVN-Kubernetes. The subnet for the distributed transit switch that enables east-west traffic. This subnet cannot overlap with any other subnets used by OVN-Kubernetes or on the host itself. It must be large enough to accommodate one IP address per node in your cluster. The default value is **fd97::/64**.
internalJoinSubnet	string	If your existing network infrastructure overlaps with the **fd98::/64** IPv6 subnet, you can specify a different IP address range for internal use by OVN-Kubernetes. You must ensure that the IP address range does not overlap with any other subnet used by your OpenShift Container Platform installation. The IP address range must be larger than the maximum number of nodes that can be added to the cluster. The default value is **fd98::/64**.

Table 6.6. **policyAuditConfig** object

Field	Type	Description
rateLimit	integer	The maximum number of messages to generate every second per node. The default value is **20** messages per second.
maxFileSize	integer	The maximum size for the audit log in bytes. The default value is **50000000** or 50 MB.
maxLogFiles	integer	The maximum number of log files that are retained.
destination	string	One of the following additional audit log targets: **libc** The libc **syslog()** function of the journald process on the host. **udp:<host>:<port>** A syslog server. Replace **<host>:<port>** with the host and port of the syslog server. **unix:<file>** A Unix Domain Socket file specified by **<file>**. **null** Do not send the audit logs to any additional target.

Field	Type	Description
syslogFacility	string	The syslog facility, such as **kern**, as defined by RFC5424. The default value is **local0**.

Table 6.7. **gatewayConfig** object

Field	Type	Description
routingViaHost	**boolean**	Set this field to **true** to send egress traffic from pods to the host networking stack. For highly-specialized installations and applications that rely on manually configured routes in the kernel routing table, you might want to route egress traffic to the host networking stack. By default, egress traffic is processed in OVN to exit the cluster and is not affected by specialized routes in the kernel routing table. The default value is **false**. This field has an interaction with the Open vSwitch hardware offloading feature. If you set this field to **true**, you do not receive the performance benefits of the offloading because egress traffic is processed by the host networking stack.
ipForwarding	**object**	You can control IP forwarding for all traffic on OVN-Kubernetes managed interfaces by using the **ipForwarding** specification in the **Network** resource. Specify **Restricted** to only allow IP forwarding for Kubernetes related traffic. Specify **Global** to allow forwarding of all IP traffic. For new installations, the default is **Restricted**. For updates to OpenShift Container Platform 4.14 or later, the default is **Global**. NOTE The default value of **Restricted** sets the IP forwarding to drop.
ipv4	**object**	Optional: Specify an object to configure the internal OVN-Kubernetes masquerade address for host to service traffic for IPv4 addresses.
ipv6	**object**	Optional: Specify an object to configure the internal OVN-Kubernetes masquerade address for host to service traffic for IPv6 addresses.

Table 6.8. **gatewayConfig.ipv4** object

Field	Type	Description

Field	Type	Description
internalMasqueradeSubnet	**string**	The masquerade IPv4 addresses that are used internally to enable host to service traffic. The host is configured with these IP addresses as well as the shared gateway bridge interface. The default value is **169.254.169.0/29**. IMPORTANT For OpenShift Container Platform 4.17 and later versions, clusters use **169.254.0.0/17** as the default masquerade subnet. For upgraded clusters, there is no change to the default masquerade subnet.

Table 6.9. **gatewayConfig.ipv6** object

Field	Type	Description
internalMasqueradeSubnet	**string**	The masquerade IPv6 addresses that are used internally to enable host to service traffic. The host is configured with these IP addresses as well as the shared gateway bridge interface. The default value is **fd69::/125**. IMPORTANT For OpenShift Container Platform 4.17 and later versions, clusters use **fd69::/112** as the default masquerade subnet. For upgraded clusters, there is no change to the default masquerade subnet.

Table 6.10. **ipsecConfig** object

Field	Type	Description
mode	**string**	Specifies the behavior of the IPsec implementation. Must be one of the following values: • **Disabled**: IPsec is not enabled on cluster nodes. • **External**: IPsec is enabled for network traffic with external hosts. • **Full**: IPsec is enabled for pod traffic and network traffic with external hosts.

NOTE

You can only change the configuration for your cluster network plugin during cluster installation, except for the **gatewayConfig** field that can be changed at runtime as a postinstallation activity.

Example OVN-Kubernetes configuration with IPSec enabled

```
defaultNetwork:
  type: OVNKubernetes
  ovnKubernetesConfig:
    mtu: 1400
    genevePort: 6081
    ipsecConfig:
      mode: Full
```

6.6.4. Cluster Network Operator example configuration

A complete CNO configuration is specified in the following example:

Example Cluster Network Operator object

```
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
  networkType: OVNKubernetes
```

6.7. ADDITIONAL RESOURCES

- **Network** API in the **operator.openshift.io** API group
- Expanding the cluster network IP address range
- How to configure OVN to use kernel routing table

CHAPTER 7. DNS OPERATOR IN OPENSHIFT CONTAINER PLATFORM

In OpenShift Container Platform, the DNS Operator deploys and manages a CoreDNS instance to provide a name resolution service to pods inside the cluster, enables DNS-based Kubernetes Service discovery, and resolves internal **cluster.local** names.

7.1. CHECKING THE STATUS OF THE DNS OPERATOR

You can check the DNS Operator deployment and Cluster Operator status. The DNS Operator is deployed during installation with a **Deployment** object.

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

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   MESSAGE
dns      4.1.15-0.11  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, and the DNS service has a cluster IP address.

7.2. VIEW THE DEFAULT DNS

View the default DNS resource and cluster DNS settings to verify the DNS configuration or troubleshoot DNS issues.

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
  Cluster IP:      172.30.0.10
...
```

where:

Status.Cluster Domain

Specifiecs the base DNS domain used to construct fully qualified pod and service domain names.

Status.Cluster IP

Specifies the address that pods query for name resolution. The IP is defined as the 10th address in the service CIDR range.

2. To find the service CIDR range, such as **172.30.0.0/16**, of your cluster, use the **oc get** command:

```
$ oc get networks.config/cluster -o jsonpath='{$.status.serviceNetwork}'
```

7.3. USING DNS FORWARDING

Configure DNS forwarding servers and upstream resolvers for the cluster.

You can use DNS forwarding to override the default forwarding configuration in the **/etc/resolv.conf** file in the following ways:

- Specify name servers (**spec.servers**) for every 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.
- Provide a list of upstream DNS servers (**spec.upstreamResolvers**).
- Change the default forwarding policy.

NOTE

A DNS forwarding configuration for the default domain can have both the default servers specified in the **/etc/resolv.conf** file and the upstream DNS servers.

Procedure

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

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

After you issue the previous command, the Operator creates and updates the config map named **dns-default** with additional server configuration blocks based on **spec.servers**. If none of the servers have a zone that matches the query, then name resolution falls back to the upstream DNS servers.

Configuring DNS forwarding

```
apiVersion: operator.openshift.io/v1
kind: DNS
metadata:
  name: default
spec:
  cache:
    negativeTTL: 0s
    positiveTTL: 0s
  logLevel: Normal
  nodePlacement: {}
  operatorLogLevel: Normal
  servers:
  - name: example-server
    zones:
    - example.com
    forwardPlugin:
      policy: Random
      upstreams:
      - 1.1.1.1
      - 2.2.2.2:5353
  upstreamResolvers:
    policy: Random
    protocolStrategy: ""
    transportConfig: {}
    upstreams:
    - type: SystemResolvConf
    - type: Network
      address: 1.2.3.4
      port: 53
    status:
      clusterDomain: cluster.local
      clusterIP: x.y.z.10
      conditions:
...
```

where:

spec.servers.name

Must comply with the **rfc6335** service name syntax.

spec.servers.zones

Must conform to the **rfc1123** subdomain syntax. The cluster domain **cluster.local** is invalid for **zones**.

spec.servers.forwardPlugin.policy

Specifies the upstream selection policy. Defaults to **Random**; allowed values are **RoundRobin** and **Sequential**.

spec.servers.forwardPlugin.upstreams

Must provide no more than 15 **upstreams** entries per **forwardPlugin**.

spec.upstreamResolvers.upstreams

Specifies an **upstreamResolvers** to override the default forwarding policy and forward DNS resolution to the specified DNS resolvers (upstream resolvers) for the default domain. You can use this field when you need custom upstream resolvers; otherwise queries use the servers declared in **/etc/resolv.conf**.

spec.upstreamResolvers.policy

Specifies the upstream selection order. Defaults to **Sequential**; allowed values are **Random**, **RoundRobin**, and **Sequential**.

spec.upstreamResolvers.protocolStrategy

Specify **TCP** to force the protocol to use for upstream DNS requests, even if the request uses UDP. Valid values are **TCP** and omitted. When omitted, the platform chooses a default, normally the protocol of the original client request.

spec.upstreamResolvers.transportConfig

Specifies the transport type, server name, and optional custom CA or CA bundle to use when forwarding DNS requests to an upstream resolver.

spec.upstreamResolvers.upstreams.type

Specifies two types of **upstreams**: **SystemResolvConf** or **Network**. **SystemResolvConf** configures the upstream to use **/etc/resolv.conf** and **Network** defines a **Networkresolver**. You can specify one or both.

spec.upstreamResolvers.upstreams.address

Specifies a valid IPv4 or IPv6 address when type is **Network**.

spec.upstreamResolvers.upstreams.port

Specifies an optional field to provide a port number. Valid values are between **1** and **65535**; defaults to 853 when omitted.

Additional resources

- CoreDNS forward documentation

7.4. CHECKING DNS OPERATOR STATUS

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

Procedure

- View the status of the DNS Operator:

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

Though the messages and spelling might vary in a specific release, the expected status output looks like:

```
Status:
```

```
Conditions:
  Last Transition Time:  <date>
  Message:               DNS "default" is available.
  Reason:                AsExpected
  Status:                True
  Type:                  Available
  Last Transition Time:  <date>
  Message:               Desired and current number of DNSes are equal
  Reason:                AsExpected
  Status:                False
  Type:                  Progressing
  Last Transition Time:  <date>
  Reason:                DNSNotDegraded
  Status:                False
  Type:                  Degraded
  Last Transition Time:  <date>
  Message:               DNS default is upgradeable: DNS Operator can be upgraded
  Reason:                DNSUpgradeable
  Status:                True
  Type:                  Upgradeable
```

7.5. VIEWING DNS OPERATOR LOGS

You can view DNS Operator logs to troubleshoot DNS issues, verify configuration changes, and monitor activity by using the by using the **oc logs** command.

Procedure

- View the logs of the DNS Operator by running the following command:

```
$ oc logs -n openshift-dns-operator deployment/dns-operator -c dns-operator
```

7.6. SETTING THE COREDNS LOG LEVEL

Set CoreDNS log levels to control the detail of DNS error logging.

Log levels for CoreDNS and the CoreDNS Operator are set by using different methods. You can configure the CoreDNS log level to determine the amount of detail in logged error messages. The valid values for CoreDNS log level are **Normal**, **Debug**, and **Trace**. The default **logLevel** is **Normal**.

> **NOTE**
>
> The CoreDNS error log level is always enabled. The following log level settings report different error responses:
>
> - **logLevel**: **Normal** enables the "errors" class: **log . { class error }**.
> - **logLevel**: **Debug** enables the "denial" class: **log . { class denial error }**.
> - **logLevel**: **Trace** enables the "all" class: **log . { class all }**.

Procedure

- To set **logLevel** to **Debug**, enter the following command:

```
$ oc patch dnses.operator.openshift.io/default -p '{"spec":{"logLevel":"Debug"}}' --type=merge
```

- To set **logLevel** to **Trace**, enter the following command:

```
$ oc patch dnses.operator.openshift.io/default -p '{"spec":{"logLevel":"Trace"}}' --type=merge
```

Verification

- To ensure the desired log level was set, check the config map:

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

For example, after setting the **logLevel** to **Trace**, you should see this stanza in each server block:

```
errors
log . {
    class all
}
```

7.7. VIEWING THE COREDNS LOGS

You can view CoreDNS pod logs to troubleshoot DNS issues by using the **oc logs** command.

Procedure

- View the logs of a specific CoreDNS pod by entering the following command:

```
$ oc -n openshift-dns logs -c dns <core_dns_pod_name>
```

- Follow the logs of all CoreDNS pods by entering the following command:

```
$ oc -n openshift-dns logs -c dns -l dns.operator.openshift.io/daemonset-dns=default -f --
max-log-requests=<number> (1)
```

  - **<number>**: Specifies the number of DNS pods to stream logs from. The maximum is 6.

7.8. SETTING THE COREDNS OPERATOR LOG LEVEL

You can configure the Operator log level to quickly track down OpenShift DNS issues.

The valid values for **operatorLogLevel** are **Normal**, **Debug**, and **Trace**. **Trace** has the most detailed information. The default **operatorlogLevel** is **Normal**. There are seven logging levels for Operator issues: Trace, Debug, Info, Warning, Error, Unrecoverable, and Panic. After the logging level is set, log entries with that severity or anything above it will be logged.

- **operatorLogLevel: "Normal"** sets **logrus.SetLogLevel("Info")**.
- **operatorLogLevel: "Debug"** sets **logrus.SetLogLevel("Debug")**.

- **operatorLogLevel: "Trace"** sets **logrus.SetLogLevel("Trace")**.

Procedure

- To set **operatorLogLevel** to **Debug**, enter the following command:

```
$ oc patch dnses.operator.openshift.io/default -p '{"spec":{"operatorLogLevel":"Debug"}}' --type=merge
```

- To set **operatorLogLevel** to **Trace**, enter the following command:

```
$ oc patch dnses.operator.openshift.io/default -p '{"spec":{"operatorLogLevel":"Trace"}}' --type=merge
```

Verification

1. To review the resulting change, enter the following command:

```
$ oc get dnses.operator -A -oyaml
```

You should see two log level entries. The **operatorLogLevel** applies to OpenShift DNS Operator issues, and the **logLevel** applies to the daemonset of CoreDNS pods:

```
logLevel: Trace
operatorLogLevel: Debug
```

2. To review the logs for the daemonset, enter the following command:

```
$ oc logs -n openshift-dns ds/dns-default
```

7.9. TUNING THE COREDNS CACHE

For CoreDNS, you can configure the maximum duration of both successful or unsuccessful caching, also known respectively as positive or negative caching. Tuning the cache duration of DNS query responses can reduce the load for any upstream DNS resolvers.

> **WARNING**
>
> Setting TTL fields to low values could lead to an increased load on the cluster, any upstream resolvers, or both.

Procedure

1. Edit the DNS Operator object named **default** by running the following command:

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

2. Modify the time-to-live (TTL) caching values:

Configuring DNS caching

```
apiVersion: operator.openshift.io/v1
kind: DNS
metadata:
  name: default
spec:
  cache:
    positiveTTL: 1h
    negativeTTL: 0.5h10m
```

where:

spec.cache.positiveTTL

Specifies a string value that is converted to its respective number of seconds by CoreDNS. If this field is omitted, the value is assumed to be **0s** and the cluster uses the internal default value of **900s** as a fallback.

spec.cache.negativeTTL

Specifies a string value that is converted to its respective number of seconds by CoreDNS. If this field is omitted, the value is assumed to be **0s** and the cluster uses the internal default value of **30s** as a fallback.

Verification

1. To review the change, look at the config map again by running the following command:

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

2. Verify that you see entries that look like the following example:

```
       cache 3600 {
            denial 9984 2400
        }
```

Additional resources

- CoreDNS cache

7.9.1. Changing the DNS Operator managementState

You can change from the default **Managed** state to **Unmanaged** to stop the DNS Operator from managing its resources in order to apply a workaround or test a configuration change.

The DNS Operator manages the CoreDNS component to provide a name resolution service for pods and services in the cluster. The **managementState** of the DNS Operator is set to **Managed** by default, which means that the DNS Operator is actively managing its resources. You can change it to **Unmanaged**, which means the DNS Operator is not managing its resources.

The following are use cases for changing the DNS Operator **managementState**:

- You are a developer and want to test a configuration change to see if it fixes an issue in CoreDNS. You can stop the DNS Operator from overwriting the configuration change by setting the **managementState** to **Unmanaged**.

- You are a cluster administrator and have reported an issue with CoreDNS, but need to apply a workaround until the issue is fixed. You can set the **managementState** field of the DNS Operator to **Unmanaged** to apply the workaround.

> NOTE
>
> You cannot upgrade while the **managementState** is set to **Unmanaged**.

Procedure

1. Change **managementState** to **Unmanaged** in the DNS Operator by running the following command:

```
oc patch dns.operator.openshift.io default --type merge --patch '{"spec":
{"managementState":"Unmanaged"}}'
```

2. Review **managementState** of the DNS Operator by using the **jsonpath** command-line JSON parser:

```
$ oc get dns.operator.openshift.io default -ojsonpath='{.spec.managementState}'
```

7.9.2. Controlling DNS pod placement

Control where CoreDNS and node-resolver pods run by using taints, tolerations, and selectors.

The DNS Operator has two daemon sets: one for CoreDNS called **dns-default** and one for managing the **/etc/hosts** file called **node-resolver**.

You can assign and run CoreDNS pods on specified nodes. For example, if the cluster administrator has configured security policies that prohibit communication between pairs of nodes, you can configure CoreDNS pods to run on a restricted set of nodes.

DNS service is available to all pods if the following circumstances are true:

- DNS pods are running on some nodes in the cluster.

- The nodes on which DNS pods are not running have network connectivity to nodes on which DNS pods are running,

The **node-resolver** daemon set must run on every node host because it adds an entry for the cluster image registry to support pulling images. The **node-resolver** pods have only one job: to look up the **image-registry.openshift-image-registry.svc** service's cluster IP address and add it to **/etc/hosts** on the node host so that the container runtime can resolve the service name.

As a cluster administrator, you can use a custom node selector to configure the daemon set for CoreDNS to run or not run on certain nodes.

Prerequisites

- You installed the **oc** CLI.

- You are logged in to the cluster as a user with **cluster-admin** privileges.
- Your DNS Operator **managementState** is set to **Managed**.

Procedure

- To allow the daemon set for CoreDNS to run on certain nodes, configure a taint and toleration:

1. Set a taint on the nodes that you want to control DNS pod placement by entering the following command:

```
$ oc adm taint nodes <node_name> dns-only=abc:NoExecute
```

   - Replace **<node_name>** with the actual name of the node.

2. Modify the DNS Operator object named **default** to include the corresponding toleration by entering the following command:

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

3. Specify a taint key and a toleration for the taint. The following toleration matches the taint set on the nodes.

```
apiVersion: operator.openshift.io/v1
kind: DNS
metadata:
  name: default
spec:
  nodePlacement:
    tolerations:
    - effect: NoExecute
      key: "dns-only"
      operator: Equal
      value: abc
      tolerationSeconds: 3600
```

   - If the **key** field is set to **dns-only**, it can be tolerated indefinitely.
   - The **tolerationSeconds** field is optional.

4. Optional: To specify node placement using a node selector, modify the default DNS Operator:

   a. Edit the DNS Operator object named **default** to include a node selector:

```
apiVersion: operator.openshift.io/v1
kind: DNS
metadata:
  name: default
spec:
  nodePlacement:
    nodeSelector:
      node-role.kubernetes.io/control-plane: ""
```

- The **spec.nodePlacement.nodeSelector** field in the example ensures that the CoreDNS pods run only on control plane nodes.

7.9.3. Configuring DNS forwarding with TLS

Configure DNS forwarding with TLS to secure queries to upstream resolvers.

When working in a highly regulated environment, you might need the ability to secure DNS traffic when forwarding requests to upstream resolvers so that you can ensure additional DNS traffic and data privacy.

Be aware that CoreDNS caches forwarded connections for 10 seconds. CoreDNS will hold a TCP connection open for those 10 seconds if no request is issued.

> **NOTE**
>
> With large clusters, ensure that your DNS server is aware that it might get many new connections to hold open because you can initiate a connection per node. Set up your DNS hierarchy accordingly to avoid performance issues.

Procedure

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

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

Cluster administrators can configure transport layer security (TLS) for forwarded DNS queries.

Configuring DNS forwarding with TLS

```
apiVersion: operator.openshift.io/v1
kind: DNS
metadata:
  name: default
spec:
  servers:
  - name: example_server
    zones:
    - example.com
    forwardPlugin:
      transportConfig:
        transport: TLS
        tls:
          caBundle:
            name: mycacert
          serverName: dnstls.example.com
      policy: Random
      upstreams:
      - 1.1.1.1
      - 2.2.2.2:5353
  upstreamResolvers:
    transportConfig:
      transport: TLS
      tls:
        caBundle:
```

```
        name: mycacert
      serverName: dnstls.example.com
  upstreams:
  - type: Network
    address: 1.2.3.4
    port: 53
```

where:

spec.servers.name

Must comply with the **rfc6335** service name syntax.

spec.servers.zones

Must conform to the **rfc1123** subdomain syntax. The cluster domain, **cluster.local**, is invalid for **zones**.

spec.servers.forwardPlugin.transportConfig.transport

Must be set to **TLS** when configuring TLS forwarding.

spec.servers.forwardPlugin.transportConfig.tls.serverName

Must be set to the server name indication (SNI) server name used to validate the upstream TLS certificate.

spec.servers.forwardPlugin.policy

Specifies the upstream selection policy. Defaults to **Random**; valid values are **RoundRobin** and **Sequential**.

spec.servers.forwardPlugin.upstreams

Must provide upstream resolvers; maximum 15 entries per **forwardPlugin**.

spec.upstreamResolvers.upstreams

Specifies an optional field to override the default policy for the default domain. Use the **Network** type only when TLS is enabled and provide an IP address. If omitted, queries use **/etc/resolv.conf**.

spec.upstreamResolvers.upstreams.address

Must be a valid IPv4 or IPv6 address.

spec.upstreamResolvers.upstreams.port

Specifies an optional field to provide a port number. Valid values are between **1** and **65535**; defaults to 853 when omitted.

NOTE

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

Verification

1. View the config map:

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

Sample DNS ConfigMap based on TLS forwarding example

```yaml
apiVersion: v1
data:
  Corefile: |
    example.com:5353 {
        forward . 1.1.1.1 2.2.2.2:5353
    }
    bar.com:5353 example.com:5353 {
      forward . 3.3.3.3 4.4.4.4:5454
    }
    .:5353 {
        errors
        health
        kubernetes cluster.local in-addr.arpa ip6.arpa {
            pods insecure
            upstream
            fallthrough in-addr.arpa ip6.arpa
        }
        prometheus :9153
        forward . /etc/resolv.conf 1.2.3.4:53 {
            policy Random
        }
        cache 30
        reload
    }
kind: ConfigMap
metadata:
  labels:
    dns.operator.openshift.io/owning-dns: default
  name: dns-default
  namespace: openshift-dns
```

- The **data.Corefile** key contains the Corefile configuration for the DNS server. Changes to the **forwardPlugin** triggers a rolling update of the CoreDNS daemon set.

Additional resources

- CoreDNS forward documentation

CHAPTER 8. 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.

8.1. OPENSHIFT CONTAINER PLATFORM INGRESS OPERATOR

When you create your OpenShift Container Platform cluster, pods and services running on the cluster are each allocated their own IP addresses. The IP addresses are accessible to other pods and services running nearby but are not accessible to outside clients.

The Ingress Operator makes it possible for external clients to access your service 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. Configurations within the Ingress Controller, such as the ability to define **endpointPublishingStrategy** type and internal load balancing, provide ways to publish Ingress Controller endpoints.

8.2. 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.

8.3. INGRESS CONTROLLER CONFIGURATION PARAMETERS

The **IngressController** custom resource (CR) includes optional configuration parameters that you can configure to meet specific needs for your organization.

<table>
<tr><th>Parameter</th><th>Description</th></tr>
<tr><td>domain</td><td>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.</td></tr>
<tr><td>replicas</td><td>replicas is the number of Ingress Controller replicas. If not set, the default value is 2.</td></tr>
</table>

Parameter	Description
endpointPublishingStrategy	**endpointPublishingStrategy** is used to publish the Ingress Controller endpoints to other networks, enable load balancer integrations, and provide access to other systems. For cloud environments, use the **loadBalancer** field to configure the endpoint publishing strategy for your Ingress Controller. On Google Cloud, AWS, and Azure you can configure the following **endpointPublishingStrategy** fields: • **loadBalancer.scope** • **loadBalancer.allowedSourceRanges** If not set, the default value is based on **infrastructure.config.openshift.io/cluster .status.platform**: • Azure: **LoadBalancerService** (with External scope) • Google Cloud: **LoadBalancerService** (with External scope) For most platforms, the **endpointPublishingStrategy** value can be updated. On Google Cloud, you can configure the following **endpointPublishingStrategy** fields: • **loadBalancer.scope** • **loadbalancer.providerParameters.gcp.clientAccess** For non-cloud environments, such as a bare-metal platform, use the **NodePortService**, **HostNetwork**, or **Private** fields to configure the endpoint publishing strategy for your Ingress Controller. If you do not set a value in one of these fields, the default value is based on binding ports specified in the **.status.platform** value in the **IngressController** CR. If you need to update the **endpointPublishingStrategy** value after your cluster is deployed, you can configure the following **endpointPublishingStrategy** fields: • **hostNetwork.protocol** • **nodePort.protocol** • **private.protocol**

<table>
<tr><th>Parameter</th><th>Description</th></tr>
<tr><td>defaultCertificate</td><td>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.</td></tr>
<tr><td>namespaceSelector</td><td>namespaceSelector is used to filter the set of namespaces serviced by the Ingress Controller. This is useful for implementing shards.</td></tr>
<tr><td>routeSelector</td><td>routeSelector is used to filter the set of Routes serviced by the Ingress Controller. This is useful for implementing shards.</td></tr>
<tr><td>nodePlacement</td><td>nodePlacement enables explicit control over the scheduling of the Ingress Controller.

If not set, the defaults values are used.

NOTE

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

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

</td></tr>
</table>

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 **1.3**. NOTE Ciphers and the minimum TLS version of the configured security profile are reflected in the **TLSProfile** status. IMPORTANT The Ingress Operator converts the TLS **1.0** of an **Old** or **Custom** profile to **1.1**.
clientTLS	**clientTLS** authenticates client access to the cluster and services; as a result, mutual TLS authentication is enabled. If not set, then client TLS is not enabled. **clientTLS** has the required subfields, **spec.clientTLS.clientCertificatePolicy** and **spec.clientTLS.ClientCA**. The **ClientCertificatePolicy** subfield accepts one of the two values: **Required** or **Optional**. The **ClientCA** subfield specifies a config map that is in the openshift-config namespace. The config map should contain a CA certificate bundle. The **AllowedSubjectPatterns** is an optional value that specifies a list of regular expressions, which are matched against the distinguished name on a valid client certificate to filter requests. The regular expressions must use PCRE syntax. At least one pattern must match a client certificate's distinguished name; otherwise, the Ingress Controller rejects the certificate and denies the connection. If not specified, the Ingress Controller does not reject certificates based on the distinguished name.

<table>
<tr><th>Parameter</th><th>Description</th></tr>
<tr><td>routeAdmission</td><td> routeAdmission defines a policy for handling new route claims, such as allowing or denying claims across namespaces.
namespaceOwnership describes how hostname claims across namespaces should be handled. The default is Strict.

Strict: does not allow routes to claim the same hostname across namespaces.
InterNamespaceAllowed: allows routes to claim different paths of the same hostname 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.
</td></tr>
</table>

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.
	      • **maxLength** is the maximum length of the syslog message. It must be between **480** and **4096** bytes. If this field is empty, the maximum length is set to the default value of **1024** bytes.
	      • **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.

<table>
<tr><th>Parameter</th><th>Description</th></tr>
<tr><td>httpHeaders</td><td>
httpHeaders defines the policy for HTTP headers.
By setting the forwardedHeaderPolicy for the IngressControllerHTTPHeaders, you specify when and how the Ingress Controller sets the Forwarded, X-Forwarded-For, X-Forwarded-Host, X-Forwarded-Port, X-Forwarded-Proto, and X-Forwarded-Proto-Version HTTP headers.
By default, the policy is set to Append.

Append specifies that the Ingress Controller appends the headers, preserving any existing headers.
Replace specifies that the Ingress Controller sets the headers, removing any existing headers.
IfNone specifies that the Ingress Controller sets the headers if they are not already set.
Never specifies that the Ingress Controller never sets the headers, preserving any existing headers.

By setting headerNameCaseAdjustments, you can specify case adjustments that can be applied to HTTP header names. Each adjustment is specified as an HTTP header name with the desired capitalization. For example, specifying X-Forwarded-For indicates that the x-forwarded-for HTTP header should be adjusted to have the specified capitalization.
These adjustments are only applied to cleartext, edge-terminated, and re-encrypt routes, and only when using HTTP/1.
For request headers, these adjustments are applied only for routes that have the haproxy.router.openshift.io/h1-adjust-case=true annotation. For response headers, these adjustments are applied to all HTTP responses. If this field is empty, no request headers are adjusted.
actions specifies options for performing certain actions on headers. Headers cannot be set or deleted for TLS passthrough connections. The actions field has additional subfields spec.httpHeader.actions.response and spec.httpHeader.actions.request:

The response subfield specifies a list of HTTP response headers to set or delete.
The request subfield specifies a list of HTTP request headers to set or delete.

</td></tr>
</table>

<table>
<tr><th>Parameter</th><th>Description</th></tr>
<tr><td>httpCompression</td><td>httpCompression defines the policy for HTTP traffic compression.
mimeTypes defines a list of MIME types to which compression should be applied. For example, text/css; charset=utf-8, text/html, text/*, image/svg+xml, application/octet-stream, X-custom/customsub, using the format pattern,type/subtype; [;attribute=value]. The types are: application, image, message, multipart, text, video, or a custom type prefaced by X-; e.g. To see the full notation for MIME types and subtypes, see RFC1341</td></tr>
<tr><td>httpErrorCodePages</td><td>httpErrorCodePages specifies custom HTTP error code response pages. By default, an IngressController uses error pages built into the IngressController image.</td></tr>
<tr><td>httpCaptureCookies</td><td>httpCaptureCookies specifies HTTP cookies that you want to capture in access logs. If the httpCaptureCookies field is empty, the access logs do not capture the cookies.

For any cookie that you want to capture, the following parameters must be in your IngressController configuration:
name specifies the name of the cookie.
maxLength specifies tha maximum length of the cookie.
matchType specifies if the field name of the cookie exactly matches the capture cookie setting or is a prefix of the capture cookie setting. The matchType field uses the Exact and Prefix parameters.
For example:<pre>httpCaptureCookies:
- matchType: Exact
  maxLength: 128
  name: MYCOOKIE</pre></td></tr>
</table>

Parameter	Description

httpCaptureHeaders	**httpCaptureHeaders** specifies the HTTP headers that you want to capture in the access logs. If the **httpCaptureHeaders** field is empty, the access logs do not capture the headers.
	
	httpCaptureHeaders contains two lists of headers to capture in the access logs. The two lists of header fields are **request** and **response**. In both lists, the **name** field must specify the header name and the**maxlength** field must specify the maximum length of the header. For example:
	
	`httpCaptureHeaders:`
	`  request:`
	`  - maxLength: 256`
	`    name: Connection`
	`  - maxLength: 128`
	`    name: User-Agent`
	`  response:`
	`  - maxLength: 256`
	`    name: Content-Type`
	`  - maxLength: 256`
	`    name: Content-Length`

tuningOptions	**tuningOptions** specifies options for tuning the performance of Ingress Controller pods. • **clientFinTimeout** specifies how long a connection is held open while waiting for the client response to the server closing the connection. The default timeout is **1s**. • **clientTimeout** specifies how long a connection is held open while waiting for a client response. The default timeout is **30s**. • **headerBufferBytes** specifies how much memory is reserved, in bytes, for Ingress Controller connection sessions. This value must be at least **16384** if HTTP/2 is enabled for the Ingress Controller. If not set, the default value is **32768** bytes. Setting this field not recommended because **headerBufferBytes** values that are too small can break the Ingress Controller, and **headerBufferBytes** values that are too large could cause the Ingress Controller to use significantly more memory than necessary. • **headerBufferMaxRewriteBytes** specifies how much memory should be reserved, in bytes, from **headerBufferBytes** for HTTP header rewriting and appending for Ingress Controller connection sessions. The minimum value for **headerBufferMaxRewriteBytes** is **4096**. **headerBufferBytes** must be greater than **headerBufferMaxRewriteBytes** for incoming HTTP requests. If not set, the default value is **8192** bytes. Setting this field not recommended because **headerBufferMaxRewriteBytes** values that are too small can break the Ingress Controller and **headerBufferMaxRewriteBytes** values that are too large could cause the Ingress Controller to use significantly more memory than necessary. • **healthCheckInterval** specifies how long the router waits between health checks. The default is **5s**.

Parameter	Description
	serverFinTimeout specifies how long a connection is held open while waiting for the server response to the client that is closing the connection. The default timeout is **1s**. **serverTimeout** specifies how long a connection is held open while waiting for a server response. The default timeout is **30s**. **threadCount** specifies the number of threads to create per HAProxy process. Creating more threads allows each Ingress Controller pod to handle more connections, at the cost of more system resources being used. HAProxy supports up to **64** threads. If this field is empty, the Ingress Controller uses the default value of **4** threads. The default value can change in future releases. Setting this field is not recommended because increasing the number of HAProxy threads allows Ingress Controller pods to use more CPU time under load, and prevent other pods from receiving the CPU resources they need to perform. Reducing the number of threads can cause the Ingress Controller to perform poorly. **tlsInspectDelay** specifies how long the router can hold data to find a matching route. Setting this value too short can cause the router to fall back to the default certificate for edge-terminated, reencrypted, or passthrough routes, even when using a better matched certificate. The default inspect delay is **5s**. **tunnelTimeout** specifies how long a tunnel connection, including websockets, remains open while the tunnel is idle. The default timeout is **1h**. **maxConnections** specifies the maximum number of simultaneous connections that can be established per HAProxy process. Increasing this value allows each ingress controller pod to handle more connections at the cost of additional system resources. Permitted values are **0**, **-1**, any value within the range **2000** and **2000000**, or the field can be left empty. - If this field is left empty or has the value **0**, the Ingress Controller will use the default value of **50000**. This value is subject to change in future releases. - If the field has the value of **-1**, then HAProxy will dynamically compute a maximum value based on the available **ulimits** in the running container. This process results in a large computed value that will incur significant memory usage compared to the current default value of **50000**. - If the field has a value that is greater than the current operating system limit, the HAProxy process will not start. - If you choose a discrete value and the router pod is migrated to a new node, it is possible the new node does not have an identical **ulimit** configured. In such cases, the pod fails to start. - If you have nodes with different **ulimits** configured, and you choose a discrete value, it is recommended to use the value of **-1** for this field so that the maximum number of connections is calculated at runtime.

Parameter	Description
logEmptyRequests	**logEmptyRequests** specifies connections for which no request is received and logged. These empty requests come from load balancer health probes or web browser speculative connections (preconnect) and logging these requests can be undesirable. However, these requests can be caused by network errors, in which case logging empty requests can be useful for diagnosing the errors. These requests can be caused by port scans, and logging empty requests can aid in detecting intrusion attempts. Allowed values for this field are **Log** and **Ignore**. The default value is **Log**. The **LoggingPolicy** type accepts either one of two values: • **Log**: Setting this value to **Log** indicates that an event should be logged. • **Ignore**: Setting this value to **Ignore** sets the **dontlognull** option in the HAproxy configuration.
HTTPEmptyRequestsPolicy	**HTTPEmptyRequestsPolicy** describes how HTTP connections are handled if the connection times out before a request is received. Allowed values for this field are **Respond** and **Ignore**. The default value is **Respond**. The **HTTPEmptyRequestsPolicy** type accepts either one of two values: • **Respond**: If the field is set to **Respond**, the Ingress Controller sends an HTTP **400** or **408** response, logs the connection if access logging is enabled, and counts the connection in the appropriate metrics. • **Ignore**: Setting this option to **Ignore** adds the **http-ignore-probes** parameter in the HAproxy configuration. If the field is set to **Ignore**, the Ingress Controller closes the connection without sending a response, then logs the connection, or incrementing metrics. These connections come from load balancer health probes or web browser speculative connections (preconnect) and can be safely ignored. However, these requests can be caused by network errors, so setting this field to **Ignore** can impede detection and diagnosis of problems. These requests can be caused by port scans, in which case logging empty requests can aid in detecting intrusion attempts.

8.3.1. Ingress Controller TLS security profiles

TLS security profiles provide a way for servers to regulate which ciphers a connecting client can use when connecting to the server.

8.3.1.1. Understanding TLS security profiles

You can use a TLS (Transport Layer Security) security profile, as described in this section, to define which TLS ciphers are required by various OpenShift Container Platform components.

The OpenShift Container Platform TLS security profiles are based on Mozilla recommended configurations.

You can specify one of the following TLS security profiles for each component:

Table 8.1. TLS security profiles

Profile	Description
Old	This profile is intended for use with legacy clients or libraries. The profile is based on the Old backward compatibility recommended configuration. The **Old** profile requires a minimum TLS version of 1.0. NOTE For the Ingress Controller, the minimum TLS version is converted from 1.0 to 1.1.
Intermediate	This profile is the default TLS security profile for the Ingress Controller, kubelet, and control plane. The profile is based on the Intermediate compatibility recommended configuration. The **Intermediate** profile requires a minimum TLS version of 1.2. NOTE This profile is the recommended configuration for the majority of clients.
Modern	This profile is intended for use with modern clients that have no need for backwards compatibility. This profile is based on the Modern compatibility recommended configuration. The **Modern** profile requires a minimum TLS version of 1.3.
Custom	This profile allows you to define the TLS version and ciphers to use. WARNING Use caution when using a **Custom** profile, because invalid configurations can cause problems.

> **NOTE**
>
> When using one of the predefined 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 might cause a new profile configuration to be applied, resulting in a rollout.

8.3.1.2. Configuring the TLS security profile for the Ingress Controller

To configure a TLS security profile for an Ingress Controller, edit the **IngressController** custom resource (CR) to specify a predefined or custom TLS security profile. If a TLS security profile is not configured, the default value is based on the TLS security profile set for the API server.

Sample IngressController CR that configures the Old TLS security profile

```
apiVersion: operator.openshift.io/v1
kind: IngressController
 ...
spec:
  tlsSecurityProfile:
    old: {}
    type: Old
 ...
```

The TLS security profile defines the minimum TLS version and the TLS ciphers for TLS connections for Ingress Controllers.

You can see the ciphers and the minimum TLS version of the configured TLS security profile in the **IngressController** custom resource (CR) under **Status.Tls Profile** and the configured TLS security profile under **Spec.Tls Security Profile**. For the **Custom** TLS security profile, the specific ciphers and minimum TLS version are listed under both parameters.

> **NOTE**
>
> The HAProxy Ingress Controller image supports TLS **1.3** and the **Modern** profile.
>
> The Ingress Operator also converts the TLS **1.0** of an **Old** or **Custom** profile to **1.1**.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.

Procedure

1. Edit the **IngressController** CR in the **openshift-ingress-operator** project to configure the TLS security profile:

```
$ oc edit IngressController default -n openshift-ingress-operator
```

2. Add the **spec.tlsSecurityProfile** field:

Sample IngressController CR for a Custom profile

```
apiVersion: operator.openshift.io/v1
kind: IngressController
 ...
spec:
  tlsSecurityProfile:
    type: Custom (1)
    custom: (2)
      ciphers: (3)
      - ECDHE-ECDSA-CHACHA20-POLY1305
      - ECDHE-RSA-CHACHA20-POLY1305
      - ECDHE-RSA-AES128-GCM-SHA256
      - ECDHE-ECDSA-AES128-GCM-SHA256
      minTLSVersion: VersionTLS11
 ...
```

1 1 1 1 Specify the TLS security profile type (**Old**, **Intermediate**, or **Custom**). The default is **Intermediate**.

2 Specify the appropriate field for the selected type:

- **old: {}**
- **intermediate: {}**
- **custom:**

3 For the **custom** type, specify a list of TLS ciphers and minimum accepted TLS version.

3. Save the file to apply the changes.

Verification

- Verify that the profile is set in the **IngressController** CR:

```
$ oc describe IngressController default -n openshift-ingress-operator
```

Example output

```
Name:         default
Namespace:    openshift-ingress-operator
Labels:       <none>
Annotations:  <none>
API Version:  operator.openshift.io/v1
Kind:         IngressController
 ...
Spec:
 ...
  Tls Security Profile:
    Custom:
      Ciphers:
        ECDHE-ECDSA-CHACHA20-POLY1305
        ECDHE-RSA-CHACHA20-POLY1305
        ECDHE-RSA-AES128-GCM-SHA256
```

```
      ECDHE-ECDSA-AES128-GCM-SHA256
    Min TLS Version:  VersionTLS11
  Type:            Custom
...
```

8.3.1.3. Configuring mutual TLS authentication

You can configure the Ingress Controller to enable mutual TLS (mTLS) authentication by setting a **spec.clientTLS** value. The **clientTLS** value configures the Ingress Controller to verify client certificates. This configuration includes setting a **clientCA** value, which is a reference to a config map. The config map contains the PEM-encoded CA certificate bundle that is used to verify a client's certificate. Optionally, you can also configure a list of certificate subject filters.

If the **clientCA** value specifies an X509v3 certificate revocation list (CRL) distribution point, the Ingress Operator downloads and manages a CRL config map based on the HTTP URI X509v3 **CRL Distribution Point** specified in each provided certificate. The Ingress Controller uses this config map during mTLS/TLS negotiation. Requests that do not provide valid certificates are rejected.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- You have a PEM-encoded CA certificate bundle.
- If your CA bundle references a CRL distribution point, you must have also included the end-entity or leaf certificate to the client CA bundle. This certificate must have included an HTTP URI under **CRL Distribution Points**, as described in RFC 5280. For example:

  ```
  Issuer: C=US, O=Example Inc, CN=Example Global G2 TLS RSA SHA256 2020 CA1
           Subject: SOME SIGNED CERT            X509v3 CRL Distribution Points:
                  Full Name:
                    URI:http://crl.example.com/example.crl
  ```

Procedure

1. In the **openshift-config** namespace, create a config map from your CA bundle:

   ```
   $ oc create configmap \
     router-ca-certs-default \
     --from-file=ca-bundle.pem=client-ca.crt \ (1)
     -n openshift-config
   ```

   (1) The config map data key must be **ca-bundle.pem**, and the data value must be a CA certificate in PEM format.

2. Edit the **IngressController** resource in the **openshift-ingress-operator** project:

   ```
   $ oc edit IngressController default -n openshift-ingress-operator
   ```

3. Add the **spec.clientTLS** field and subfields to configure mutual TLS:

   Sample IngressController CR for a clientTLS profile that specifies filtering patterns

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  clientTLS:
    clientCertificatePolicy: Required
    clientCA:
      name: router-ca-certs-default
    allowedSubjectPatterns:
    - "^/CN=example.com/ST=NC/C=US/O=Security/OU=OpenShift$"
```

4. Optional, get the Distinguished Name (DN) for **allowedSubjectPatterns** by entering the following command.

```
$ openssl x509 -in custom-cert.pem -noout -subject
```

Example output

```
subject=C=US, ST=NC, O=Security, OU=OpenShift, CN=example.com
```

8.4. 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
```

8.5. 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
```

8.6. VIEW INGRESS CONTROLLER LOGS

You can view your Ingress Controller logs.

Procedure

- View your Ingress Controller logs:

```
$ oc logs --namespace=openshift-ingress-operator deployments/ingress-operator -c
<container_name>
```

8.7. 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>
```

8.8. CREATING A CUSTOM INGRESS CONTROLLER

As a cluster administrator, you can create a new custom Ingress Controller. Because the default Ingress Controller might change during OpenShift Container Platform updates, creating a custom Ingress Controller can be helpful when maintaining a configuration manually that persists across cluster updates.

This example provides a minimal spec for a custom Ingress Controller. To further customize your custom Ingress Controller, see "Configuring the Ingress Controller".

Prerequisites

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

Procedure

1. Create a YAML file that defines the custom **IngressController** object:

Example custom-ingress-controller.yaml file

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: <custom_name> (1)
  namespace: openshift-ingress-operator
spec:
  defaultCertificate:
    name: <custom-ingress-custom-certs> (2)
  replicas: 1 (3)
  domain: <custom_domain> (4)
```

(1) Specify the a custom **name** for the **IngressController** object.

(2) Specify the name of the secret with the custom wildcard certificate.

(3) Minimum replica needs to be ONE

4 Specify the domain to your domain name. The domain specified on the IngressController object and the domain used for the certificate must match. For example, if the domain

2. Create the object by running the following command:

```
$ oc create -f custom-ingress-controller.yaml
```

8.9. CONFIGURING THE INGRESS CONTROLLER

8.9.1. 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 meets the following requirements:
  - The certificate is valid for the ingress domain.
  - The certificate uses the **subjectAltName** extension to specify a wildcard domain, such as ***.apps.ocp4.example.com**.
- You must have an **IngressController** CR, which includes just having the **default IngressController** CR. You can run the following command to check that you have an **IngressController** CR:

```
$ oc --namespace openshift-ingress-operator get ingresscontrollers
```

NOTE

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

Procedure

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

NOTE

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

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

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

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

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

3. Verify the update was effective:

   ```
   $ echo Q |\
     openssl s_client -connect console-openshift-console.apps.<domain>:443 -showcerts 2>/dev/null |\
     openssl x509 -noout -subject -issuer -enddate
   ```

   where:

   <domain>

   Specifies the base domain name for your cluster.

   #### Example output

   ```
   subject=C = US, ST = NC, L = Raleigh, O = RH, OU = OCP4, CN = *.apps.example.com
   issuer=C = US, ST = NC, L = Raleigh, O = RH, OU = OCP4, CN = example.com
   notAfter=May 10 08:32:45 2022 GM
   ```

   #### TIP

   You can alternatively apply the following YAML to set a custom default certificate:

   ```
   apiVersion: operator.openshift.io/v1
   kind: IngressController
   metadata:
     name: default
     namespace: openshift-ingress-operator
   spec:
     defaultCertificate:
       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.

8.9.2. Removing a custom default certificate

As an administrator, you can remove a custom certificate that you configured an Ingress Controller to use.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- You have installed the OpenShift CLI (**oc**).
- You previously configured a custom default certificate for the Ingress Controller.

Procedure

- To remove the custom certificate and restore the certificate that ships with OpenShift Container Platform, enter the following command:

```
$ oc patch -n openshift-ingress-operator ingresscontrollers/default \
  --type json -p $'- op: remove\n  path: /spec/defaultCertificate'
```

  There can be a delay while the cluster reconciles the new certificate configuration.

Verification

- To confirm that the original cluster certificate is restored, enter the following command:

```
$ echo Q | \
  openssl s_client -connect console-openshift-console.apps.<domain>:443 -showcerts
2>/dev/null | \
  openssl x509 -noout -subject -issuer -enddate
```

  where:

  <domain>
  Specifies the base domain name for your cluster.

  #### Example output

```
subject=CN = *.apps.<domain>
issuer=CN = ingress-operator@1620633373
notAfter=May 10 10:44:36 2023 GMT
```

8.9.3. Autoscaling an Ingress Controller

You can automatically scale an Ingress Controller to dynamically meet routing performance or availability requirements. For example, the requirement to increase throughput.

The following procedure provides an example for scaling up the default Ingress Controller.

Prerequisites

- You have the OpenShift CLI (**oc**) installed.
- You have access to an OpenShift Container Platform cluster as a user with the **cluster-admin** role.

- On VMware vSphere, bare-metal, and Nutanix installer-provisioned infrastructure, scaling up Ingress Controller pods does not improve external traffic performance. To improve performance, ensure that you complete the following prerequisites:
  - You manually configured a user-managed load balancer for your cluster.
  - You ensured that the load balancer was configured for the cluster nodes that handle incoming traffic from the Ingress Controller.
- You installed the Custom Metrics Autoscaler Operator and an associated KEDA Controller.
  - You can install the Operator by using OperatorHub on the web console. After you install the Operator, you can create an instance of **KedaController**.

Procedure

1. Create a service account to authenticate with Thanos by running the following command:

```
$ oc create -n openshift-ingress-operator serviceaccount thanos && oc describe -n openshift-ingress-operator serviceaccount thanos
```

Example output

```
Name:                thanos
Namespace:           openshift-ingress-operator
Labels:              <none>
Annotations:         <none>
Image pull secrets:  thanos-dockercfg-kfvf2
Mountable secrets:   thanos-dockercfg-kfvf2
Tokens:              <none>
Events:              <none>
```

2. Manually create the service account secret token with the following command:

```
$ oc apply -f - <<EOF
apiVersion: v1
kind: Secret
metadata:
  name: thanos-token
  namespace: openshift-ingress-operator
  annotations:
    kubernetes.io/service-account.name: thanos
type: kubernetes.io/service-account-token
EOF
```

3. Define a **TriggerAuthentication** object within the **openshift-ingress-operator** namespace by using the service account’s token.

   a. Create the **TriggerAuthentication** object and pass the value of the **secret** variable to the **TOKEN** parameter:

```
$ oc apply -f - <<EOF
apiVersion: keda.sh/v1alpha1
kind: TriggerAuthentication
metadata:
```

```
  name: keda-trigger-auth-prometheus
  namespace: openshift-ingress-operator
spec:
  secretTargetRef:
  - parameter: bearerToken
    name: thanos-token
    key: token
  - parameter: ca
    name: thanos-token
    key: ca.crt
EOF
```

4. Create and apply a role for reading metrics from Thanos:

   a. Create a new role, **thanos-metrics-reader.yaml**, that reads metrics from pods and nodes:

      thanos-metrics-reader.yaml

      ```
      apiVersion: rbac.authorization.k8s.io/v1
      kind: Role
      metadata:
        name: thanos-metrics-reader
        namespace: openshift-ingress-operator
      rules:
      - apiGroups:
        - ""
        resources:
        - pods
        - nodes
        verbs:
        - get
      - apiGroups:
        - metrics.k8s.io
        resources:
        - pods
        - nodes
        verbs:
        - get
        - list
        - watch
      - apiGroups:
        - ""
        resources:
        - namespaces
        verbs:
        - get
      ```

   b. Apply the new role by running the following command:

      ```
      $ oc apply -f thanos-metrics-reader.yaml
      ```

5. Add the new role to the service account by entering the following commands:

   ```
   $ oc adm policy -n openshift-ingress-operator add-role-to-user thanos-metrics-reader -z
   thanos --role-namespace=openshift-ingress-operator
   ```

```
$ oc adm policy -n openshift-ingress-operator add-cluster-role-to-user cluster-monitoring-view
-z thanos
```

> **NOTE**
>
> The argument **add-cluster-role-to-user** is only required if you use cross-namespace queries. The following step uses a query from the **kube-metrics** namespace which requires this argument.

6. Create a new **ScaledObject** YAML file, **ingress-autoscaler.yaml**, that targets the default Ingress Controller deployment:

Example ScaledObject definition

```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: ingress-scaler
  namespace: openshift-ingress-operator
spec:
  scaleTargetRef: (1)
    apiVersion: operator.openshift.io/v1
    kind: IngressController
    name: default
    envSourceContainerName: ingress-operator
  minReplicaCount: 1
  maxReplicaCount: 20 (2)
  cooldownPeriod: 1
  pollingInterval: 1
  triggers:
  - type: prometheus
    metricType: AverageValue
    metadata:
      serverAddress: https://thanos-querier.openshift-monitoring.svc.cluster.local:9091 (3)
      namespace: openshift-ingress-operator (4)
      metricName: 'kube-node-role'
      threshold: '1'
      query: 'sum(kube_node_role{role="worker",service="kube-state-metrics"})' (5)
      authModes: "bearer"
    authenticationRef:
      name: keda-trigger-auth-prometheus
```

1. The custom resource that you are targeting. In this case, the Ingress Controller.
2. Optional: The maximum number of replicas. If you omit this field, the default maximum is set to 100 replicas.
3. The Thanos service endpoint in the **openshift-monitoring** namespace.
4. The Ingress Operator namespace.
5. This expression evaluates to however many worker nodes are present in the deployed cluster.

> **IMPORTANT**
>
> If you are using cross-namespace queries, you must target port 9091 and not port 9092 in the **serverAddress** field. You also must have elevated privileges to read metrics from this port.

7. Apply the custom resource definition by running the following command:

```
$ oc apply -f ingress-autoscaler.yaml
```

Verification

- Verify that the default Ingress Controller is scaled out to match the value returned by the **kube-state-metrics** query by running the following commands:
  - Use the **grep** command to search the Ingress Controller YAML file for the number of replicas:

    ```
    $ oc get -n openshift-ingress-operator ingresscontroller/default -o yaml | grep replicas:
    ```

  - Get the pods in the **openshift-ingress** project:

    ```
    $ oc get pods -n openshift-ingress
    ```

    Example output

    ```
    NAME                           READY   STATUS    RESTARTS   AGE
    router-default-7b5df44ff-l9pmm   2/2     Running   0          17h
    router-default-7b5df44ff-s5sl5   2/2     Running   0          3d22h
    router-default-7b5df44ff-wwsth   2/2     Running   0          66s
    ```

Additional resources

- Installing the custom metrics autoscaler
- Enabling monitoring for user-defined projects
- Understanding custom metrics autoscaler trigger authentications
- Understanding custom metrics autoscaler triggers
- Understanding how to add custom metrics autoscalers

8.9.4. 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**.

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

Prerequisites

- On VMware vSphere, bare-metal, and Nutanix installer-provisioned infrastructure, scaling up Ingress Controller pods does not improve external traffic performance. To improve performance, ensure that you complete the following prerequisites:
  - You manually configured a user-managed load balancer for your cluster.
  - You ensured that the load balancer was configured for the cluster nodes that handle incoming traffic from the Ingress Controller.

Procedure

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

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

2. Scale the default **IngressController** to the desired number of replicas by 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
```

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}'
```

TIP

You can alternatively apply the following YAML to scale an Ingress Controller to three replicas:

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  replicas: 3   (1)
```

(1) If you need a different amount of replicas, change the **replicas** value.

8.9.5. 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 OpenShift 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
  logging:
    access:
      destination:
        type: Container
```

- 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.address** 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
  logging:
    access:
      destination:
        type: Syslog
        syslog:
          address: 1.2.3.4
          port: 10514
```

> **NOTE**
>
> The **syslog** destination port must be UDP.
>
> The **syslog** destination address must be an IP address. It does not support DNS hostname.

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
  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
logging:
  access: null
```

Allow the Ingress Controller to modify the HAProxy log length when using a sidecar.

- Use **spec.logging.access.destination.syslog.maxLength** if you are using **spec.logging.access.destination.type: Syslog**.

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  replicas: 2
  logging:
    access:
      destination:
        type: Syslog
        syslog:
          address: 1.2.3.4
          maxLength: 4096
          port: 10514
```

- Use **spec.logging.access.destination.container.maxLength** if you are using **spec.logging.access.destination.type: Container**.

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  replicas: 2
  logging:
    access:
      destination:
        container:
          maxLength: 8192
        type: Container
      httpCaptureHeaders:
        request:
        - maxLength: 128
          name: X-Forwarded-For
```

- To view the original client source IP address by using the **X-Forwarded-For** header in the **Ingress** access logs, see the "Capturing Original Client IP from the X-Forwarded-For Header in Ingress and Application Logs" Red Hat Knowledgebase solution.

Additional resources

- Capturing Original Client IP from the X-Forwarded-For Header in Ingress and Application Logs

8.9.6. Setting Ingress Controller thread count

A cluster administrator can set the thread count to increase the amount of incoming connections a cluster can handle. You can patch an existing Ingress Controller to increase the amount of threads.

Prerequisites

- The following assumes that you already created an Ingress Controller.

Procedure

- Update the Ingress Controller to increase the number of threads:

```
$ oc -n openshift-ingress-operator patch ingresscontroller/default --type=merge -p '{"spec":
{"tuningOptions": {"threadCount": 8}}}'
```

> **NOTE**
>
> If you have a node that is capable of running large amounts of resources, you can configure **spec.nodePlacement.nodeSelector** with labels that match the capacity of the intended node, and configure **spec.tuningOptions.threadCount** to an appropriately high value.

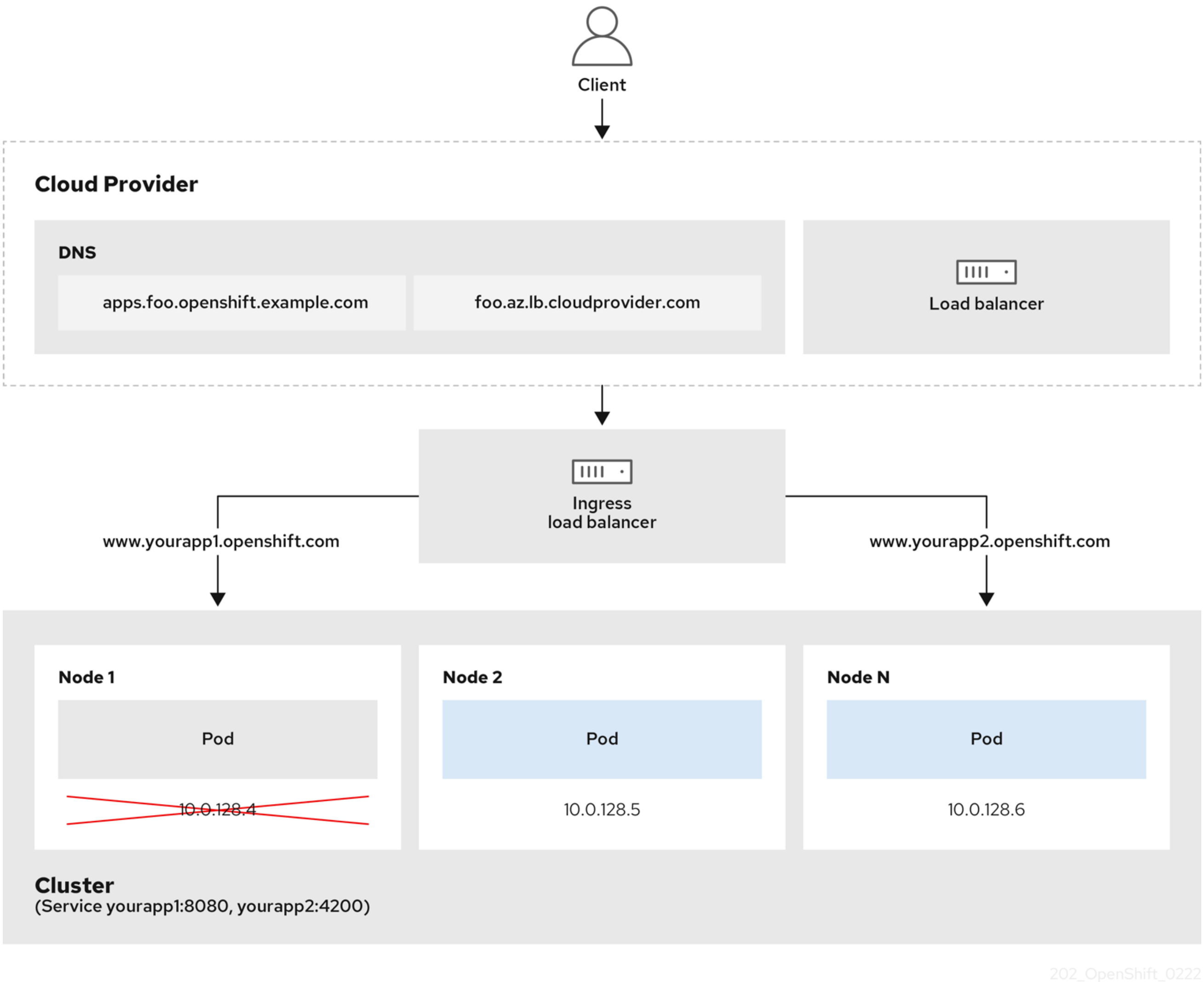
8.9.7. Configuring an Ingress Controller to use an internal load balancer

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

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

> **IMPORTANT**
>
> If you want to change the **scope** for an **IngressController**, you can change the **.spec.endpointPublishingStrategy.loadBalancer.scope** parameter after the custom resource (CR) is created.

Figure 8.1. Diagram of LoadBalancer



The preceding graphic shows the following concepts pertaining to OpenShift Container Platform Ingress LoadBalancerService endpoint publishing strategy:

- You can load balance externally, using the cloud provider load balancer, or internally, using the OpenShift Ingress Controller Load Balancer.
- You can use the single IP address of the load balancer and more familiar ports, such as 8080 and 4200 as shown on the cluster depicted in the graphic.
- Traffic from the external load balancer is directed at the pods, and managed by the load balancer, as depicted in the instance of a down node. 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
```

8.9.8. Configuring global access for an Ingress Controller on Google Cloud

An Ingress Controller created on Google Cloud with an internal load balancer generates an internal IP address for the service. A cluster administrator can specify the global access option, which enables clients in any region within the same VPC network and compute region as the load balancer, to reach the workloads running on your cluster.

For more information, see the Google Cloud documentation for global access.

Prerequisites

- You deployed an OpenShift Container Platform cluster on Google Cloud infrastructure.
- You configured an Ingress Controller to use an internal load balancer.
- You installed the OpenShift CLI (**oc**).

Procedure

1. Configure the Ingress Controller resource to allow global access.

> **NOTE**
>
> You can also create an Ingress Controller and specify the global access option.

a. Configure the Ingress Controller resource:

```
$ oc -n openshift-ingress-operator edit ingresscontroller/default
```

b. Edit the YAML file:

Sample clientAccess configuration to Global

```
spec:
  endpointPublishingStrategy:
    loadBalancer:
      providerParameters:
        gcp:
          clientAccess: Global 1
        type: GCP
      scope: Internal
    type: LoadBalancerService
```

1 Set **gcp.clientAccess** to **Global**.

c. Save the file to apply the changes.

2. Run the following command to verify that the service allows global access:

```
$ oc -n openshift-ingress edit svc/router-default -o yaml
```

The output shows that global access is enabled for Google Cloud with the annotation, **networking.gke.io/internal-load-balancer-allow-global-access**.

8.9.9. Setting the Ingress Controller health check interval

A cluster administrator can set the health check interval to define how long the router waits between two consecutive health checks. This value is applied globally as a default for all routes. The default value is 5 seconds.

Prerequisites

- The following assumes that you already created an Ingress Controller.

Procedure

- Update the Ingress Controller to change the interval between back end health checks:

```
$ oc -n openshift-ingress-operator patch ingresscontroller/default --type=merge -p '{"spec":
{"tuningOptions": {"healthCheckInterval": "8s"}}}'
```

NOTE

To override the **healthCheckInterval** for a single route, use the route annotation **router.openshift.io/haproxy.health.check.interval**

8.9.10. Configuring the default Ingress Controller for your cluster to be internal

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

WARNING

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

IMPORTANT

If you want to change the **scope** for an **IngressController**, you can 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
```

8.9.11. 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 hostname.

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

TIP

You can alternatively apply the following YAML to configure the route admission policy:

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  routeAdmission:
    namespaceOwnership: InterNamespaceAllowed
```

8.9.12. 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
   ```

8.9.13. HTTP header configuration

To customize request and response headers for your applications, configure the Ingress Controller or apply specific route annotations. Understanding the interaction between these configuration methods ensures you effectively manage global and route-specific header policies.

You can also set certain headers by using route annotations. The various ways of configuring headers can present challenges when working together.

> **NOTE**
>
> You can only set or delete headers within an **IngressController** or **Route** CR, you cannot append them. If an HTTP header is set with a value, that value must be complete and not require appending in the future. In situations where it makes sense to append a header, such as the X-Forwarded-For header, use the **spec.httpHeaders.forwardedHeaderPolicy** field, instead of **spec.httpHeaders.actions**.

Order of precedence

When the same HTTP header is modified both in the Ingress Controller and in a route, HAProxy prioritizes the actions in certain ways depending on whether it is a request or response header.

- For HTTP response headers, actions specified in the Ingress Controller are executed after the actions specified in a route. This means that the actions specified in the Ingress Controller take precedence.
- For HTTP request headers, actions specified in a route are executed after the actions specified in the Ingress Controller. This means that the actions specified in the route take precedence.

For example, a cluster administrator sets the X-Frame-Options response header with the value **DENY** in the Ingress Controller using the following configuration:

Example IngressController spec

```
apiVersion: operator.openshift.io/v1
```

```
kind: IngressController
# ...
spec:
  httpHeaders:
    actions:
      response:
      - name: X-Frame-Options
        action:
          type: Set
          set:
            value: DENY
```

A route owner sets the same response header that the cluster administrator set in the Ingress Controller, but with the value **SAMEORIGIN** using the following configuration:

Example Route spec

```
apiVersion: route.openshift.io/v1
kind: Route
# ...
spec:
  httpHeaders:
    actions:
      response:
      - name: X-Frame-Options
        action:
          type: Set
          set:
            value: SAMEORIGIN
```

When both the **IngressController** spec and **Route** spec are configuring the X-Frame-Options response header, then the value set for this header at the global level in the Ingress Controller takes precedence, even if a specific route allows frames. For a request header, the **Route** spec value overrides the **IngressController** spec value.

This prioritization occurs because the **haproxy.config** file uses the following logic, where the Ingress Controller is considered the front end and individual routes are considered the back end. The header value **DENY** applied to the front end configurations overrides the same header with the value **SAMEORIGIN** that is set in the back end:

```
frontend public
  http-response set-header X-Frame-Options 'DENY'

frontend fe_sni
  http-response set-header X-Frame-Options 'DENY'

frontend fe_no_sni
  http-response set-header X-Frame-Options 'DENY'

backend be_secure:openshift-monitoring:alertmanager-main
  http-response set-header X-Frame-Options 'SAMEORIGIN'
```

Additionally, any actions defined in either the Ingress Controller or a route override values set using route annotations.

Special case headers

The following headers are either prevented entirely from being set or deleted, or allowed under specific circumstances:

Table 8.2. Special case header configuration options

Header name	Configurable using **IngressController** spec	Configurable using **Route** spec	Reason for disallowment	Configurable using another method
proxy	No	No	The **proxy** HTTP request header can be used to exploit vulnerable CGI applications by injecting the header value into the **HTTP_PROXY** environment variable. The **proxy** HTTP request header is also non-standard and prone to error during configuration.	No
host	No	Yes	When the **host** HTTP request header is set using the **IngressController** CR, HAProxy can fail when looking up the correct route.	No
strict-transport-security	No	No	The **strict-transport-security** HTTP response header is already handled using route annotations and does not need a separate implementation.	Yes: the **haproxy.router.openshift.io/hsts_header** route annotation

Header name	Configurable using **IngressController** spec	Configurable using **Route** spec	Reason for disallowment	Configurable using another method
cookie and **set-cookie**	No	No	The cookies that HAProxy sets are used for session tracking to map client connections to particular back-end servers. Allowing these headers to be set could interfere with HAProxy's session affinity and restrict HAProxy's ownership of a cookie.	Yes: • the **haproxy.router.openshift.io/disable_cookie** route annotation • the **haproxy.router.openshift.io/cookie_name** route annotation

8.9.14. Setting or deleting HTTP request and response headers in an Ingress Controller

You can set or delete certain HTTP request and response headers for compliance purposes or other reasons. You can set or delete these headers either for all routes served by an Ingress Controller or for specific routes.

For example, you might want to migrate an application running on your cluster to use mutual TLS, which requires that your application checks for an X-Forwarded-Client-Cert request header, but the OpenShift Container Platform default Ingress Controller provides an X-SSL-Client-Der request header.

The following procedure modifies the Ingress Controller to set the X-Forwarded-Client-Cert request header, and delete the X-SSL-Client-Der request header.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have access to an OpenShift Container Platform cluster as a user with the **cluster-admin** role.

Procedure

1. Edit the Ingress Controller resource:

```
$ oc -n openshift-ingress-operator edit ingresscontroller/default
```

2. Replace the X-SSL-Client-Der HTTP request header with the X-Forwarded-Client-Cert HTTP request header:

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  httpHeaders:
    actions: 1
      request: 2
      - name: X-Forwarded-Client-Cert 3
        action:
          type: Set 4
          set:
           value: "%{+Q}[ssl_c_der,base64]" 5
      - name: X-SSL-Client-Der
        action:
          type: Delete
```

1 The list of actions you want to perform on the HTTP headers.

2 The type of header you want to change. In this case, a request header.

3 The name of the header you want to change. For a list of available headers you can set or delete, see *HTTP header configuration*.

4 The type of action being taken on the header. This field can have the value **Set** or **Delete**.

5 When setting HTTP headers, you must provide a **value**. The value can be a string from a list of available directives for that header, for example **DENY**, or it can be a dynamic value that will be interpreted using HAProxy's dynamic value syntax. In this case, a dynamic value is added.

NOTE

For setting dynamic header values for HTTP responses, allowed sample fetchers are **res.hdr** and **ssl_c_der**. For setting dynamic header values for HTTP requests, allowed sample fetchers are **req.hdr** and **ssl_c_der**. Both request and response dynamic values can use the **lower** and **base64** converters.

3. Save the file to apply the changes.

8.9.15. Using X-Forwarded headers

You configure the HAProxy Ingress Controller to specify a policy for how to handle HTTP headers including **Forwarded** and **X-Forwarded-For**. The Ingress Operator uses the **HTTPHeaders** field to configure the **ROUTER_SET_FORWARDED_HEADERS** environment variable of the Ingress Controller.

Procedure

1. Configure the **HTTPHeaders** field for the Ingress Controller.

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

   ```
   $ oc edit IngressController
   ```

   b. Under **spec**, set the **HTTPHeaders** policy field to **Append**, **Replace**, **IfNone**, or **Never**:

   ```
   apiVersion: operator.openshift.io/v1
   kind: IngressController
   metadata:
     name: default
     namespace: openshift-ingress-operator
   spec:
     httpHeaders:
       forwardedHeaderPolicy: Append
   ```

8.9.15.1. Example use cases

As a cluster administrator, you can:

- Configure an external proxy that injects the **X-Forwarded-For** header into each request before forwarding it to an Ingress Controller.
  To configure the Ingress Controller to pass the header through unmodified, you specify the **never** policy. The Ingress Controller then never sets the headers, and applications receive only the headers that the external proxy provides.

- Configure the Ingress Controller to pass the **X-Forwarded-For** header that your external proxy sets on external cluster requests through unmodified.
  To configure the Ingress Controller to set the **X-Forwarded-For** header on internal cluster requests, which do not go through the external proxy, specify the **if-none** policy. If an HTTP request already has the header set through the external proxy, then the Ingress Controller preserves it. If the header is absent because the request did not come through the proxy, then the Ingress Controller adds the header.

As an application developer, you can:

- Configure an application-specific external proxy that injects the **X-Forwarded-For** header.
  To configure an Ingress Controller to pass the header through unmodified for an application's Route, without affecting the policy for other Routes, add an annotation **haproxy.router.openshift.io/set-forwarded-headers: if-none** or **haproxy.router.openshift.io/set-forwarded-headers: never** on the Route for the application.

> **NOTE**
>
> You can set the **haproxy.router.openshift.io/set-forwarded-headers** annotation on a per route basis, independent from the globally set value for the Ingress Controller.

8.9.16. Enable or disable HTTP/2 on Ingress Controllers

You can enable or disable transparent end-to-end HTTP/2 connectivity in HAProxy. Application owners can use HTTP/2 protocol capabilities, including single connection, header compression, binary streams, and more.

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

> **NOTE**
>
> If you enable or disable HTTP/2 connectivity for an individual Ingress Controller and for the entire cluster, the HTTP/2 configuration for the Ingress Controller takes precedence over the HTTP/2 configuration for the cluster.

To enable the use of HTTP/2 for a connection from the client to an HAProxy instance, 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.

Consider the following use cases for an HTTP/2 connection for each route type:

- For a re-encrypt route, the connection from HAProxy to the application pod can use HTTP/2 if the application supports using Application-Level Protocol Negotiation (ALPN) to negotiate HTTP/2 with HAProxy. You cannot use HTTP/2 with a re-encrypt route unless the Ingress Controller has HTTP/2 enabled.
- For a passthrough route, the connection can use HTTP/2 if the application supports using ALPN to negotiate HTTP/2 with the client. You can use HTTP/2 with a passthrough route if the Ingress Controller has HTTP/2 enabled or disabled.
- For an edge-terminated secure route, the connection uses HTTP/2 if the service specifies only **appProtocol: kubernetes.io/h2c**. You can use HTTP/2 with an edge-terminated secure route if the Ingress Controller has HTTP/2 enabled or disabled.
- For an insecure route, the connection uses HTTP/2 if the service specifies only **appProtocol: kubernetes.io/h2c**. You can use HTTP/2 with an insecure route if the Ingress Controller has HTTP/2 enabled or disabled.

> **IMPORTANT**
>
> For non-passthrough routes, the Ingress Controller negotiates its connection to the application independently of the connection from the client. This means a client might connect to the Ingress Controller and negotiate HTTP/1.1. The Ingress Controller might then connect to the application, negotiate HTTP/2, and forward the request from the client HTTP/1.1 connection by using the HTTP/2 connection to the application.
>
> This sequence of events causes an issue if the client subsequently tries to upgrade its connection from HTTP/1.1 to the WebSocket protocol. Consider that if you have an application that is intending to accept WebSocket connections, and the application attempts to allow for HTTP/2 protocol negotiation, the client fails any attempt to upgrade to the WebSocket protocol.

8.9.16.1. Enabling HTTP/2

You can enable HTTP/2 on a specific Ingress Controller, or you can enable HTTP/2 for the entire cluster.

Procedure

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

```
$ oc -n openshift-ingress-operator annotate ingresscontrollers/<ingresscontroller_name>
ingress.operator.openshift.io/default-enable-http2=true (1)
```

(1) Replace **<ingresscontroller_name>** with the name of an Ingress Controller to enable HTTP/2.

- 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
```

TIP

Alternatively, you can apply the following YAML code to enable HTTP/2:

```yaml
apiVersion: config.openshift.io/v1
kind: Ingress
metadata:
  name: cluster
  annotations:
    ingress.operator.openshift.io/default-enable-http2: "true"
```

8.9.16.2. Disabling HTTP/2

You can disable HTTP/2 on a specific Ingress Controller, or you can disable HTTP/2 for the entire cluster.

Procedure

- To disable HTTP/2 on a specific Ingress Controller, enter the **oc annotate** command:

```
$ oc -n openshift-ingress-operator annotate ingresscontrollers/<ingresscontroller_name>
ingress.operator.openshift.io/default-enable-http2=false (1)
```

(1) Replace **<ingresscontroller_name>** with the name of an Ingress Controller to disable HTTP/2.

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

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

TIP

Alternatively, you can apply the following YAML code to disable HTTP/2:

```
apiVersion: config.openshift.io/v1
kind: Ingress
metadata:
  name: cluster
  annotations:
    ingress.operator.openshift.io/default-enable-http2: "false"
```

8.9.17. Configuring the PROXY protocol for an Ingress Controller

A cluster administrator can configure the PROXY protocol when an Ingress Controller uses either the **HostNetwork**, **NodePortService**, or **Private** endpoint publishing strategy types. The PROXY protocol enables the load balancer to preserve the original client addresses for connections that the Ingress Controller receives. The original client addresses are useful for logging, filtering, and injecting HTTP headers. In the default configuration, the connections that the Ingress Controller receives only contain the source address that is associated with the load balancer.

> **WARNING**
>
> The default Ingress Controller with installer-provisioned clusters on non-cloud platforms that use a Keepalived Ingress Virtual IP (VIP) do not support the PROXY protocol.

The PROXY protocol enables the load balancer to preserve the original client addresses for connections that the Ingress Controller receives. The original client addresses are useful for logging, filtering, and injecting HTTP headers. In the default configuration, the connections that the Ingress Controller receives contain only the source IP address that is associated with the load balancer.

> **IMPORTANT**
>
> For a passthrough route configuration, servers in OpenShift Container Platform clusters cannot observe the original client source IP address. If you need to know the original client source IP address, configure Ingress access logging for your Ingress Controller so that you can view the client source IP addresses.
>
> For re-encrypt and edge routes, the OpenShift Container Platform router sets the **Forwarded** and **X-Forwarded-For** headers so that application workloads check the client source IP address.
>
> For more information about Ingress access logging, see "Configuring Ingress access logging".

Configuring the PROXY protocol for an Ingress Controller is not supported when using the **LoadBalancerService** endpoint publishing strategy type. This restriction is because when OpenShift Container Platform runs in a cloud platform, and an Ingress Controller specifies that a service load balancer should be used, the Ingress Operator configures the load balancer service and enables the PROXY protocol based on the platform requirement for preserving source addresses.

> **IMPORTANT**
>
> You must configure both OpenShift Container Platform and the external load balancer to use either the PROXY protocol or TCP.

This feature is not supported in cloud deployments. This restriction is because when OpenShift Container Platform runs in a cloud platform, and an Ingress Controller specifies that a service load balancer should be used, the Ingress Operator configures the load balancer service and enables the PROXY protocol based on the platform requirement for preserving source addresses.

> **IMPORTANT**
>
> You must configure both OpenShift Container Platform and the external load balancer to either use the PROXY protocol or to use Transmission Control Protocol (TCP).

Prerequisites

- You created an Ingress Controller.

Procedure

1. Edit the Ingress Controller resource by entering the following command in your CLI:

```
$ oc -n openshift-ingress-operator edit ingresscontroller/default
```

2. Set the PROXY configuration:

   - If your Ingress Controller uses the **HostNetwork** endpoint publishing strategy type, set the **spec.endpointPublishingStrategy.hostNetwork.protocol** subfield to **PROXY**:

     Sample hostNetwork configuration to PROXY

```
# ...
  spec:
    endpointPublishingStrategy:
      hostNetwork:
        protocol: PROXY
      type: HostNetwork
# ...
```

   - If your Ingress Controller uses the **NodePortService** endpoint publishing strategy type, set the **spec.endpointPublishingStrategy.nodePort.protocol** subfield to **PROXY**:

     Sample nodePort configuration to PROXY

```
# ...
  spec:
    endpointPublishingStrategy:
      nodePort:
```

```
      protocol: PROXY
    type: NodePortService
# ...
```

- If your Ingress Controller uses the **Private** endpoint publishing strategy type, set the **spec.endpointPublishingStrategy.private.protocol** subfield to **PROXY**:

  Sample private configuration to PROXY

  ```
  # ...
    spec:
      endpointPublishingStrategy:
        private:
          protocol: PROXY
      type: Private
  # ...
  ```

Additional resources

- Configuring Ingress access logging

8.9.18. Specifying an alternative cluster domain using the appsDomain option

As a cluster administrator, you can specify an alternative to the default cluster domain for user-created routes by configuring the **appsDomain** field. The **appsDomain** field is an optional domain for OpenShift Container Platform to use instead of the default, which is specified in the **domain** field. If you specify an alternative domain, it overrides the default cluster domain for the purpose of determining the default host for a new route.

For example, you can use the DNS domain for your company as the default domain for routes and ingresses for applications running on your cluster.

Prerequisites

- You deployed an OpenShift Container Platform cluster.
- You installed the **oc** command-line interface.

Procedure

1. Configure the **appsDomain** field by specifying an alternative default domain for user-created routes.

   a. Edit the ingress **cluster** resource:

   ```
   $ oc edit ingresses.config/cluster -o yaml
   ```

   b. Edit the YAML file:

   Sample appsDomain configuration to test.example.com

   ```
   apiVersion: config.openshift.io/v1
   kind: Ingress
   ```

```
metadata:
  name: cluster
spec:
  domain: apps.example.com          1
  appsDomain: <test.example.com>      2
```

1. Specifies the default domain. You cannot modify the default domain after installation.

2. Optional: Domain for OpenShift Container Platform infrastructure to use for application routes. Instead of the default prefix, **apps**, you can use an alternative prefix like **test**.

2. Verify that an existing route contains the domain name specified in the **appsDomain** field by exposing the route and verifying the route domain change:

   NOTE

   Wait for the **openshift-apiserver** finish rolling updates before exposing the route.

   a. Expose the route by entering the following command. The command outputs **route.route.openshift.io/hello-openshift exposed** to designate exposure of the route.

   ```
   $ oc expose service hello-openshift
   ```

   b. Get a list of routes by running the following command:

   ```
   $ oc get routes
   ```

   Example output

   ```
   NAME            HOST/PORT                                    PATH   SERVICES        PORT
   TERMINATION   WILDCARD
   hello-openshift   hello_openshift-<my_project>.test.example.com
   hello-openshift   8080-tcp                None
   ```

8.9.19. Converting HTTP header case

HAProxy lowercases HTTP header names by default; for example, changing **Host: xyz.com** to **host: xyz.com**. If legacy applications are sensitive to the capitalization of HTTP header names, use the Ingress Controller **spec.httpHeaders.headerNameCaseAdjustments** API field for a solution to accommodate legacy applications until they can be fixed.

IMPORTANT

OpenShift Container Platform includes HAProxy 2.8. If you want to update to this version of the web-based load balancer, ensure that you add the **spec.httpHeaders.headerNameCaseAdjustments** section to your cluster's configuration file.

As a cluster administrator, you can convert the HTTP header case by entering the **oc patch** command or by setting the **HeaderNameCaseAdjustments** field in the Ingress Controller YAML file.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have access to the cluster as a user with the **cluster-admin** role.

Procedure

- Capitalize an HTTP header by using the **oc patch** command.
  a. Change the HTTP header from **host** to **Host** by running the following command:

```
$ oc -n openshift-ingress-operator patch ingresscontrollers/default --type=merge --patch='{"spec":{"httpHeaders":{"headerNameCaseAdjustments":["Host"]}}}'
```

  b. Create a **Route** resource YAML file so that the annotation can be applied to the application.

Example of a route named my-application

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  annotations:
    haproxy.router.openshift.io/h1-adjust-case: true (1)
  name: <application_name>
  namespace: <application_name>
# ...
```

(1) Set **haproxy.router.openshift.io/h1-adjust-case** so that the Ingress Controller can adjust the **host** request header as specified.

- Specify adjustments by configuring the **HeaderNameCaseAdjustments** field in the Ingress Controller YAML configuration file.
  a. The following example Ingress Controller YAML file adjusts the **host** header to **Host** for HTTP/1 requests to appropriately annotated routes:

Example Ingress Controller YAML

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  httpHeaders:
    headerNameCaseAdjustments:
    - Host
```

b. The following example route enables HTTP response header name case adjustments by using the **haproxy.router.openshift.io/h1-adjust-case** annotation:

Example route YAML

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  annotations:
    haproxy.router.openshift.io/h1-adjust-case: true 1
  name: my-application
  namespace: my-application
spec:
  to:
    kind: Service
    name: my-application
```

1 Set **haproxy.router.openshift.io/h1-adjust-case** to true.

8.9.20. Using router compression

You configure the HAProxy Ingress Controller to specify router compression globally for specific MIME types. You can use the **mimeTypes** variable to define the formats of MIME types to which compression is applied. The types are: application, image, message, multipart, text, video, or a custom type prefaced by "X-". To see the full notation for MIME types and subtypes, see RFC1341.

> **NOTE**
>
> Memory allocated for compression can affect the max connections. Additionally, compression of large buffers can cause latency, like heavy regex or long lists of regex.
>
> Not all MIME types benefit from compression, but HAProxy still uses resources to try to compress if instructed to. Generally, text formats, such as html, css, and js, formats benefit from compression, but formats that are already compressed, such as image, audio, and video, benefit little in exchange for the time and resources spent on compression.

Procedure

1. Configure the **httpCompression** field for the Ingress Controller.

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

   ```
   $ oc edit -n openshift-ingress-operator ingresscontrollers/default
   ```

   b. Under **spec**, set the **httpCompression** policy field to **mimeTypes** and specify a list of MIME types that should have compression applied:

   ```
   apiVersion: operator.openshift.io/v1
   kind: IngressController
   metadata:
     name: default
     namespace: openshift-ingress-operator
   spec:
   ```

```
httpCompression:
  mimeTypes:
  - "text/html"
  - "text/css; charset=utf-8"
  - "application/json"
  ...
```

8.9.21. Exposing router metrics

You can expose the HAProxy router metrics by default in Prometheus format on the default stats port, 1936. The external metrics collection and aggregation systems such as Prometheus can access the HAProxy router metrics. You can view the HAProxy router metrics in a browser in the HTML and comma separated values (CSV) format.

Prerequisites

- You configured your firewall to access the default stats port, 1936.

Procedure

1. Get the router pod name by running the following command:

   ```
   $ oc get pods -n openshift-ingress
   ```

   Example output

   ```
   NAME                          READY   STATUS    RESTARTS   AGE
   router-default-76bfffb66c-46qwp   1/1     Running   0          11h
   ```

2. Get the router’s username and password, which the router pod stores in the **/var/lib/haproxy/conf/metrics-auth/statsUsername** and **/var/lib/haproxy/conf/metrics-auth/statsPassword** files:

   a. Get the username by running the following command:

      ```
      $ oc rsh <router_pod_name> cat metrics-auth/statsUsername
      ```

   b. Get the password by running the following command:

      ```
      $ oc rsh <router_pod_name> cat metrics-auth/statsPassword
      ```

3. Get the router IP and metrics certificates by running the following command:

   ```
   $ oc describe pod <router_pod>
   ```

4. Get the raw statistics in Prometheus format by running the following command:

   ```
   $ curl -u <user>:<password> http://<router_IP>:<stats_port>/metrics
   ```

5. Access the metrics securely by running the following command:

   ```
   $ curl -u user:password https://<router_IP>:<stats_port>/metrics -k
   ```

6. Access the default stats port, 1936, by running the following command:

```
$ curl -u <user>:<password> http://<router_IP>:<stats_port>/metrics
```

Example 8.1. Example output

```
...
# HELP haproxy_backend_connections_total Total number of connections.
# TYPE haproxy_backend_connections_total gauge
haproxy_backend_connections_total{backend="http",namespace="default",route="hello-route"} 0
haproxy_backend_connections_total{backend="http",namespace="default",route="hello-route-alt"} 0
haproxy_backend_connections_total{backend="http",namespace="default",route="hello-route01"} 0
...
# HELP haproxy_exporter_server_threshold Number of servers tracked and the current threshold value.
# TYPE haproxy_exporter_server_threshold gauge
haproxy_exporter_server_threshold{type="current"} 11
haproxy_exporter_server_threshold{type="limit"} 500
...
# HELP haproxy_frontend_bytes_in_total Current total of incoming bytes.
# TYPE haproxy_frontend_bytes_in_total gauge
haproxy_frontend_bytes_in_total{frontend="fe_no_sni"} 0
haproxy_frontend_bytes_in_total{frontend="fe_sni"} 0
haproxy_frontend_bytes_in_total{frontend="public"} 119070
...
# HELP haproxy_server_bytes_in_total Current total of incoming bytes.
# TYPE haproxy_server_bytes_in_total gauge
haproxy_server_bytes_in_total{namespace="",pod="",route="",server="fe_no_sni",service=""} 0
haproxy_server_bytes_in_total{namespace="",pod="",route="",server="fe_sni",service=""} 0
haproxy_server_bytes_in_total{namespace="default",pod="docker-registry-5-nk5fz",route="docker-registry",server="10.130.0.89:5000",service="docker-registry"} 0
haproxy_server_bytes_in_total{namespace="default",pod="hello-rc-vkjqx",route="hello-route",server="10.130.0.90:8080",service="hello-svc-1"} 0
...
```

7. Launch the stats window by entering the following URL in a browser:

```
http://<user>:<password>@<router_IP>:<stats_port>
```

8. Optional: Get the stats in CSV format by entering the following URL in a browser:

```
http://<user>:<password>@<router_ip>:1936/metrics;csv
```

8.9.22. Customizing HAProxy error code response pages

As a cluster administrator, you can specify a custom error code response page for either 503, 404, or both error pages. The HAProxy router serves a 503 error page when the application pod is not running or a 404 error page when the requested URL does not exist. For example, if you customize the 503 error code response page, then the page is served when the application pod is not running, and the default 404 error code HTTP response page is served by the HAProxy router for an incorrect route or a non-existing route.

Custom error code response pages are specified in a config map then patched to the Ingress Controller. The config map keys have two available file names as follows: **error-page-503.http** and **error-page-404.http**.

Custom HTTP error code response pages must follow the HAProxy HTTP error page configuration guidelines. Here is an example of the default OpenShift Container Platform HAProxy router http 503 error code response page. You can use the default content as a template for creating your own custom page.

By default, the HAProxy router serves only a 503 error page when the application is not running or when the route is incorrect or non-existent. This default behavior is the same as the behavior on OpenShift Container Platform 4.8 and earlier. If a config map for the customization of an HTTP error code response is not provided, and you are using a custom HTTP error code response page, the router serves a default 404 or 503 error code response page.

> **NOTE**
>
> If you use the OpenShift Container Platform default 503 error code page as a template for your customizations, the headers in the file require an editor that can use CRLF line endings.

Procedure

1. Create a config map named **my-custom-error-code-pages** in the **openshift-config** namespace:

```
$ oc -n openshift-config create configmap my-custom-error-code-pages \
  --from-file=error-page-503.http \
  --from-file=error-page-404.http
```

> **IMPORTANT**
>
> If you do not specify the correct format for the custom error code response page, a router pod outage occurs. To resolve this outage, you must delete or correct the config map and delete the affected router pods so they can be recreated with the correct information.

2. Patch the Ingress Controller to reference the **my-custom-error-code-pages** config map by name:

```
$ oc patch -n openshift-ingress-operator ingresscontroller/default --patch '{"spec":
{"httpErrorCodePages":{"name":"my-custom-error-code-pages"}}}' --type=merge
```

The Ingress Operator copies the **my-custom-error-code-pages** config map from the **openshift-config** namespace to the **openshift-ingress** namespace. The Operator names the config map according to the pattern, **<your_ingresscontroller_name>-errorpages**, in the **openshift-ingress** namespace.

3. Display the copy:

```
$ oc get cm default-errorpages -n openshift-ingress
```

Example output

```
NAME                  DATA   AGE
default-errorpages     2     25s
```

1. The example config map name is **default-errorpages** because the **default** Ingress Controller custom resource (CR) was patched.

4. Confirm that the config map containing the custom error response page mounts on the router volume where the config map key is the filename that has the custom HTTP error code response:

    - For 503 custom HTTP custom error code response:

    ```
    $ oc -n openshift-ingress rsh <router_pod> cat
    /var/lib/haproxy/conf/error_code_pages/error-page-503.http
    ```

    - For 404 custom HTTP custom error code response:

    ```
    $ oc -n openshift-ingress rsh <router_pod> cat
    /var/lib/haproxy/conf/error_code_pages/error-page-404.http
    ```

Verification

Verify your custom error code HTTP response:

1. Create a test project and application:

    ```
    $ oc new-project test-ingress
    ```

    ```
    $ oc new-app django-psql-example
    ```

2. For 503 custom http error code response:

    a. Stop all the pods for the application.

    b. Run the following curl command or visit the route hostname in the browser:

    ```
    $ curl -vk <route_hostname>
    ```

3. For 404 custom http error code response:

    a. Visit a non-existent route or an incorrect route.

    b. Run the following curl command or visit the route hostname in the browser:

    ```
    $ curl -vk <route_hostname>
    ```

4. Check if the **errorfile** attribute is properly in the **haproxy.config** file:

```
$ oc -n openshift-ingress rsh <router> cat /var/lib/haproxy/conf/haproxy.config | grep errorfile
```

8.9.23. Setting the Ingress Controller maximum connections

A cluster administrator can set the maximum number of simultaneous connections for OpenShift router deployments. You can patch an existing Ingress Controller to increase the maximum number of connections.

Prerequisites

- The following assumes that you already created an Ingress Controller

Procedure

- Update the Ingress Controller to change the maximum number of connections for HAProxy:

```
$ oc -n openshift-ingress-operator patch ingresscontroller/default --type=merge -p '{"spec":
{"tuningOptions": {"maxConnections": 7500}}}'
```

> **WARNING**
>
> If you set the **spec.tuningOptions.maxConnections** value greater than the current operating system limit, the HAProxy process will not start. See the table in the "Ingress Controller configuration parameters" section for more information about this parameter.

8.10. ADDITIONAL RESOURCES

- Configuring a custom PKI

CHAPTER 9. INGRESS NODE FIREWALL OPERATOR IN OPENSHIFT CONTAINER PLATFORM

The Ingress Node Firewall Operator provides a stateless, eBPF-based firewall for managing node-level ingress traffic in OpenShift Container Platform.

9.1. INGRESS NODE FIREWALL OPERATOR

The Ingress Node Firewall Operator provides ingress firewall rules at a node level that you can specify and manage in the firewall configurations.

To deploy the daemon set created by the Operator, you create an **IngressNodeFirewallConfig** custom resource (CR). The Operator applies the **IngressNodeFirewallConfig** CR to create ingress node firewall daemon set **daemon**, which run on all nodes that match the **nodeSelector**.

You configure **rules** of the **IngressNodeFirewall** CR and apply them to clusters using the **nodeSelector** and setting values to "true".

IMPORTANT

The Ingress Node Firewall Operator supports only stateless firewall rules.

Network interface controllers (NICs) that do not support native XDP drivers will run at a lower performance.

For OpenShift Container Platform 4.14 or later, you must run Ingress Node Firewall Operator on RHEL 9.0 or later.

9.2. INSTALLING THE INGRESS NODE FIREWALL OPERATOR

As a cluster administrator, you can install the Ingress Node Firewall Operator to enable node-level ingress firewalling by using the OpenShift Container Platform CLI.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have an account with administrator privileges.

Procedure

1. To create the **openshift-ingress-node-firewall** namespace, enter the following command:

```
$ cat << EOF| oc create -f -
apiVersion: v1
kind: Namespace
metadata:
  labels:
    pod-security.kubernetes.io/enforce: privileged
    pod-security.kubernetes.io/enforce-version: v1.24
  name: openshift-ingress-node-firewall
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: ingress-node-firewall-operators
  namespace: openshift-ingress-node-firewall
EOF
```

3. Subscribe to the Ingress Node Firewall Operator.

    a. To create a **Subscription** CR for the Ingress Node Firewall Operator, enter the following command:

    ```
    $ cat << EOF| oc create -f -
    apiVersion: operators.coreos.com/v1alpha1
    kind: Subscription
    metadata:
      name: ingress-node-firewall-sub
      namespace: openshift-ingress-node-firewall
    spec:
      name: ingress-node-firewall
      channel: stable
      source: redhat-operators
      sourceNamespace: openshift-marketplace
    EOF
    ```

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

```
$ oc get ip -n openshift-ingress-node-firewall
```

Example output

```
NAME            CSV                                         APPROVAL    APPROVED
install-5cvnz   ingress-node-firewall.4.18.0-202211122336   Automatic   true
```

5. To verify the version of the Operator, enter the following command:

```
$ oc get csv -n openshift-ingress-node-firewall
```

Example output

```
NAME                                        DISPLAY                          VERSION               REPLACES
PHASE
ingress-node-firewall.4.18.0-202211122336   Ingress Node Firewall Operator   4.18.0-
202211122336   ingress-node-firewall.4.18.0-202211102047   Succeeded
```

9.3. INSTALLING THE INGRESS NODE FIREWALL OPERATOR USING THE WEB CONSOLE

As a cluster administrator, you can install the Ingress Node Firewall Operator to enable node-level ingress firewalling by using the web console.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have an account with administrator privileges.

Procedure

1. Install the Ingress Node Firewall Operator:
   a. In the OpenShift Container Platform web console, click **Operators → OperatorHub**.
   b. Select **Ingress Node Firewall Operator** from the list of available Operators, and then click **Install**.
   c. On the **Install Operator** page, under **Installed Namespace**, select **Operator recommended Namespace**.
   d. Click **Install**.
2. Verify that the Ingress Node Firewall Operator is installed successfully:
   a. Navigate to the **Operators → Installed Operators** page.
   b. Ensure that **Ingress Node Firewall Operator** is listed in the **openshift-ingress-node-firewall** project with a **Status** of **InstallSucceeded**.

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

   If the Operator does not have a **Status** of **InstallSucceeded**, troubleshoot using the following steps:

   - Inspect the **Operator Subscriptions** and **Install Plans** tabs for any failures or errors under **Status**.
   - Navigate to the **Workloads → Pods** page and check the logs for pods in the **openshift-ingress-node-firewall** project.
   - Check the namespace of the YAML file. If the annotation is missing, you can add the annotation **workload.openshift.io/allowed=management** to the Operator namespace with the following command:

     ```
     $ oc annotate ns/openshift-ingress-node-firewall
     workload.openshift.io/allowed=management
     ```

NOTE

For single-node OpenShift clusters, the **openshift-ingress-node-firewall** namespace requires the **workload.openshift.io/allowed=management** annotation.

9.4. DEPLOYING INGRESS NODE FIREWALL OPERATOR

To deploy the Ingress Node Firewall Operator, create a **IngressNodeFirewallConfig** custom resource that will deploy the Operator's daemon set. You can deploy one or multiple **IngressNodeFirewall** CRDs to nodes by applying firewall rules.

Prerequisite

- The Ingress Node Firewall Operator is installed.

Procedure

1. Create the **IngressNodeFirewallConfig** inside the **openshift-ingress-node-firewall** namespace named **ingressnodefirewallconfig**.

2. Run the following command to deploy Ingress Node Firewall Operator rules:

```
$ oc apply -f rule.yaml
```

9.5. INGRESS NODE FIREWALL CONFIGURATION OBJECT

Review configuration fields so you can define how the Operator deploys the firewall.

The fields for the Ingress Node Firewall configuration object are described in the following table:

Table 9.1. Ingress Node Firewall Configuration object

Field	Type	Description
metadata.name	**string**	The name of the CR object. The name of the firewall rules object must be **ingressnodefirewallconfig**.
metadata.namespace	**string**	Namespace for the Ingress Firewall Operator CR object. The **IngressNodeFirewallConfig** CR must be created inside the **openshift-ingress-node-firewall** namespace.

<table>
<tr><th>Field</th><th>Type</th><th>Description</th></tr>
<tr><td>spec.nodeSelector</td><td>string</td><td>A node selection constraint used to target nodes through specified node labels. For example:
<pre>
apiVersion: ingressnodefirewall.openshift.io/v1alpha1
kind: IngressNodeFirewallConfig
metadata:
  name: ingressnodefirewallconfig
  namespace: openshift-ingress-node-firewall
spec:
  nodeSelector:
    node-role.kubernetes.io/worker: ""
</pre>
NOTE
One label used in nodeSelector must match a label on the nodes in order for the daemon set to start. For example, if the node labels node-role.kubernetes.io/worker and node-type.kubernetes.io/vm are applied to a node, then at least one label must be set using nodeSelector for the daemon set to start.</td></tr>
<tr><td>spec.ebpfProgramManagerMode</td><td>boolean</td><td>Specifies if the Node Ingress Firewall Operator uses the eBPF Manager Operator or not to manage eBPF programs. This capability is a Technology Preview feature.

For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope.</td></tr>
</table>

NOTE

The Operator consumes the CR and creates an ingress node firewall daemon set on all the nodes that match the **nodeSelector**.

9.5.1. Ingress Node Firewall Operator example configuration

A complete Ingress Node Firewall Configuration is specified in the following example:

Example Ingress Node Firewall Configuration object

```
apiVersion: ingressnodefirewall.openshift.io/v1alpha1
kind: IngressNodeFirewallConfig
metadata:
  name: ingressnodefirewallconfig
  namespace: openshift-ingress-node-firewall
spec:
  nodeSelector:
    node-role.kubernetes.io/worker: ""
```

NOTE

The Operator consumes the CR and creates an ingress node firewall daemon set on all the nodes that match the **nodeSelector**.

9.5.2. Ingress Node Firewall rules object

You can review rule fields and examples to define which ingress traffic is allowed or denied by using the Ingress Node Firewall rules object.

The fields for the Ingress Node Firewall rules object are described in the following table:

Table 9.2. Ingress Node Firewall rules object

Field	Type	Description
metadata.name	**string**	The name of the CR object.
interfaces	**array**	The fields for this object specify the interfaces to apply the firewall rules to. For example, **- en0** and **- en1**.
nodeSelector	**array**	You can use **nodeSelector** to select the nodes to apply the firewall rules to. Set the value of your named **nodeselector** labels to **true** to apply the rule.
ingress	**object**	**ingress** allows you to configure the rules that allow outside access to the services on your cluster.

9.5.2.1. Ingress object configuration

The values for the **ingress** object are defined in the following table:

Table 9.3. ingress object

Field	Type	Description
sourceCIDRs	**array**	Allows you to set the CIDR block. You can configure multiple CIDRs from different address families. NOTE Different CIDRs allow you to use the same order rule. In the case that there are multiple **IngressNodeFirewall** objects for the same nodes and interfaces with overlapping CIDRs, the **order** field will specify which rule is applied first. Rules are applied in ascending order.

Field	Type	Description
rules	**array**	Ingress firewall **rules.order** objects are ordered starting at **1** for each **source.CIDR** with up to 100 rules per CIDR. Lower order rules are executed first. **rules.protocolConfig.protocol** supports the following protocols: TCP, UDP, SCTP, ICMP and ICMPv6. ICMP and ICMPv6 rules can match against ICMP and ICMPv6 types or codes. TCP, UDP, and SCTP rules can match against a single destination port or a range of ports using **<start : end-1>** format. Set **rules.action** to **allow** to apply the rule or **deny** to disallow the rule. NOTE Ingress firewall rules are verified using a verification webhook that blocks any invalid configuration. The verification webhook prevents you from blocking any critical cluster services such as the API server.

9.5.2.2. Ingress Node Firewall rules object example

A complete Ingress Node Firewall configuration is specified in the following example:

Example Ingress Node Firewall configuration

```
apiVersion: ingressnodefirewall.openshift.io/v1alpha1
kind: IngressNodeFirewall
metadata:
  name: ingressnodefirewall
spec:
  interfaces:
  - eth0
  nodeSelector:
    matchLabels:
      <label_name>: <label_value>
  ingress:
  - sourceCIDRs:
       - 172.16.0.0/12
    rules:
    - order: 10
      protocolConfig:
        protocol: ICMP
        icmp:
          icmpType: 8 #ICMP Echo request
```

```
   action: Deny
  - order: 20
   protocolConfig:
    protocol: TCP
    tcp:
     ports: "8000-9000"
   action: Deny
 - sourceCIDRs:
   - fc00:f853:ccd:e793::0/64
  rules:
  - order: 10
   protocolConfig:
    protocol: ICMPv6
    icmpv6:
     icmpType: 128 #ICMPV6 Echo request
   action: Deny
```

+ A **<label_name>** and a **<label_value>** must exist on the node and must match the **nodeselector** label and value applied to the nodes you want the **ingressfirewallconfig** CR to run on. The **<label_value>** can be **true** or **false**. By using **nodeSelector** labels, you can target separate groups of nodes to apply different rules to using the **ingressfirewallconfig** CR.

9.5.2.3. Zero trust Ingress Node Firewall rules object example

Zero trust Ingress Node Firewall rules can provide additional security to multi-interface clusters. For example, you can use zero trust Ingress Node Firewall rules to drop all traffic on a specific interface except for SSH.

A complete configuration of a zero trust Ingress Node Firewall rule for a network-interface cluster is specified in the following example:

> **IMPORTANT**
>
> Users need to add all ports their application will use to their allowlist in the following case to ensure proper functionality.

Example zero trust Ingress Node Firewall rules

```
apiVersion: ingressnodefirewall.openshift.io/v1alpha1
kind: IngressNodeFirewall
metadata:
 name: ingressnodefirewall-zero-trust
spec:
 interfaces:
 - eth1
 nodeSelector:
  matchLabels:
   <ingress_firewall_label_name>: <label_value>
 ingress:
 - sourceCIDRs:
   - 0.0.0.0/0
  rules:
  - order: 10
   protocolConfig:
```

```
      protocol: TCP
      tcp:
        ports: 22
    action: Allow
  - order: 20
    action: Deny
```

> **IMPORTANT**
>
> eBPF Manager Operator integration 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 Technology Preview Features Support Scope .

9.6. INGRESS NODE FIREWALL OPERATOR INTEGRATION

Learn when to use eBPF Manager to load and manage Ingress Node Firewall programs.

The Ingress Node Firewall uses eBPF programs to implement some of its key firewall functionality. By default these eBPF programs are loaded into the kernel using a mechanism specific to the Ingress Node Firewall. You can configure the Ingress Node Firewall Operator to use the eBPF Manager Operator for loading and managing these programs instead.

When this integration is enabled, the following limitations apply:

- The Ingress Node Firewall Operator uses TCX if XDP is not available and TCX is incompatible with bpfman.
- The Ingress Node Firewall Operator daemon set pods remain in the **ContainerCreating** state until the firewall rules are applied.
- The Ingress Node Firewall Operator daemon set pods run as privileged.

9.7. CONFIGURING INGRESS NODE FIREWALL OPERATOR TO USE THE EBPF MANAGER OPERATOR

Configure the Ingress Node Firewall to use eBPF Manager for program lifecycle control.

The Ingress Node Firewall uses eBPF programs to implement some of its key firewall functionality. By default these eBPF programs are loaded into the kernel using a mechanism specific to the Ingress Node Firewall.

As a cluster administrator, you can configure the Ingress Node Firewall Operator to use the eBPF Manager Operator for loading and managing these programs instead, adding additional security and observability functionality.

Prerequisites

- You have installed the OpenShift CLI (**oc**).

- You have an account with administrator privileges.
- You installed the Ingress Node Firewall Operator.
- You have installed the eBPF Manager Operator.

Procedure

1. Apply the following labels to the **ingress-node-firewall-system** namespace:

```
$ oc label namespace openshift-ingress-node-firewall \
    pod-security.kubernetes.io/enforce=privileged \
    pod-security.kubernetes.io/warn=privileged --overwrite
```

2. Edit the **IngressNodeFirewallConfig** object named **ingressnodefirewallconfig** and set the **ebpfProgramManagerMode** field:

Ingress Node Firewall Operator configuration object

```
apiVersion: ingressnodefirewall.openshift.io/v1alpha1
kind: IngressNodeFirewallConfig
metadata:
  name: ingressnodefirewallconfig
  namespace: openshift-ingress-node-firewall
spec:
  nodeSelector:
    node-role.kubernetes.io/worker: ""
  ebpfProgramManagerMode: <ebpf_mode>
```

where:

<ebpf_mode>: Specifies whether or not the Ingress Node Firewall Operator uses the eBPF Manager Operator to manage eBPF programs. Must be either **true** or **false**. If unset, eBPF Manager is not used.

9.8. VIEWING INGRESS NODE FIREWALL OPERATOR RULES

Inspect existing rules and configs to confirm the firewall is applied as intended.

Procedure

1. Run the following command to view all current rules :

```
$ oc get ingressnodefirewall
```

2. Choose one of the returned **<resource>** names and run the following command to view the rules or configs:

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

9.9. TROUBLESHOOTING THE INGRESS NODE FIREWALL OPERATOR

You can verify the status and view the logs to diagnose ingress firewall deployment or rule issues.

Procedure

- Run the following command to list installed Ingress Node Firewall custom resource definitions (CRD):

```
$ oc get crds | grep ingressnodefirewall
```

Example output

```
NAME               READY   UP-TO-DATE   AVAILABLE   AGE
ingressnodefirewallconfigs.ingressnodefirewall.openshift.io       2022-08-25T10:03:01Z
ingressnodefirewallnodestates.ingressnodefirewall.openshift.io    2022-08-25T10:03:00Z
ingressnodefirewalls.ingressnodefirewall.openshift.io             2022-08-25T10:03:00Z
```

- Run the following command to view the state of the Ingress Node Firewall Operator:

```
$ oc get pods -n openshift-ingress-node-firewall
```

Example output

```
NAME                                       READY  STATUS         RESTARTS  AGE
ingress-node-firewall-controller-manager   2/2    Running        0         5d21h
ingress-node-firewall-daemon-pqx56         3/3    Running        0         5d21h
```

The following fields provide information about the status of the Operator: **READY**, **STATUS**, **AGE**, and **RESTARTS**. The **STATUS** field is **Running** when the Ingress Node Firewall Operator is deploying a daemon set to the assigned nodes.

- Run the following command to collect all ingress firewall node pods' logs:

```
$ oc adm must-gather – gather_ingress_node_firewall
```

The logs are available in the sos node’s report containing eBPF **bpftool** outputs at **/sos_commands/ebpf**. These reports include lookup tables used or updated as the ingress firewall XDP handles packet processing, updates statistics, and emits events.

9.10. ADDITIONAL RESOURCES

- About the eBPF Manager Operator

CHAPTER 10. SR-IOV OPERATOR

10.1. INSTALLING THE SR-IOV NETWORK OPERATOR

To manage SR-IOV network devices and network attachments on your cluster, install the Single Root I/O Virtualization (SR-IOV) Network Operator. By using this Operator, you can centralize the configuration and lifecycle management of your SR-IOV resources.

As a cluster administrator, you can install the Single Root I/O Virtualization (SR-IOV) Network Operator by using the OpenShift Container Platform CLI or the web console.

10.1.1. Using the CLI to install the SR-IOV Network Operator

You can use the CLI to install the SR-IOV Network Operator. By using the CLI, you can deploy the Operator directly from your terminal to manage SR-IOV network devices and attachments without navigating the web console.

Prerequisites

- You installed the OpenShift CLI (**oc**).
- You have an account with **cluster-admin** privileges.
- You installed a cluster on bare-metal hardware, and you ensured that cluster nodes have hardware that supports SR-IOV.

Procedure

1. Create the **openshift-sriov-network-operator** namespace by entering the following command:

```
$ cat << EOF| oc create -f -
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-sriov-network-operator
  annotations:
    workload.openshift.io/allowed: management
EOF
```

2. Create an **OperatorGroup** custom resource (CR) by entering 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. Create a **Subscription** CR for the SR-IOV Network Operator by entering the following command:

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

4. Create an **SriovoperatorConfig** resource by entering the following command:

```
$ cat <<EOF | oc create -f -
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovOperatorConfig
metadata:
  name: default
  namespace: openshift-sriov-network-operator
spec:
  enableInjector: true
  enableOperatorWebhook: true
  logLevel: 2
  disableDrain: false
EOF
```

Verification

- To verify that the Operator is installed, enter the following command and then check that the output shows **Succeeded** for the Operator:

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

10.1.2. Using the web console to install the SR-IOV Network Operator

You can use the web console to install the SR-IOV Network Operator. By using the web console, you can deploy the Operator and manage SR-IOV network devices and attachments directly from a graphical interface without having to the CLI.

Prerequisites

- You have an account with **cluster-admin** privileges.
- You installed a cluster on bare-metal hardware, and you ensured that cluster nodes have hardware that supports SR-IOV.

Procedure

1. 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 **Installed Namespace**, select **Operator recommended Namespace**.
   d. Click **Install**.

Verification

1. Navigate to the **Operators** → **Installed Operators** page.
2. Ensure that **SR-IOV Network Operator** is listed in the **openshift-sriov-network-operator** project with a **Status** of **InstallSucceeded**.

   NOTE

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

3. If the Operator does not show as installed, complete any of the following steps to troubleshoot the issue:
   - 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.
   - Check the namespace of the YAML file. If the annotation is missing, you can add the annotation **workload.openshift.io/allowed=management** to the Operator namespace with the following command:

     ```
     $ oc annotate ns/openshift-sriov-network-operator
     workload.openshift.io/allowed=management
     ```

     NOTE

     For single-node OpenShift clusters, the annotation **workload.openshift.io/allowed=management** is required for the namespace.

10.1.3. Additional resources

- Configuring the SR-IOV Network Operator

10.2. CONFIGURING THE SR-IOV NETWORK OPERATOR

To manage SR-IOV network devices and network attachments in your cluster, use the Single Root I/O Virtualization (SR-IOV) Network Operator.

10.2.1. Configuring the SR-IOV Network Operator

To manage SR-IOV network devices and network attachments in your cluster, configure the Single Root I/O Virtualization (SR-IOV) Network Operator. You can then deploy the Operator components to your cluster.

Procedure

1. Create a **SriovOperatorConfig** custom resource (CR). The following example creates a file named **sriovOperatorConfig.yaml**:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovOperatorConfig
metadata:
  name: default
  namespace: openshift-sriov-network-operator
spec:
  disableDrain: false
  enableInjector: true
  enableOperatorWebhook: true
  logLevel: 2
  featureGates:
    metricsExporter: false
# ...
```

where:

metadata.name

Specifies the name of the SR-IOV Network Operator instance. The only valid name for the **SriovOperatorConfig** resource is **default** and the name must be in the namespace where the Operator is deployed.

spec.enableInjector

Specifies if any **network-resources-injector** pod can run in the namespace. If not specified in the CR or explicitly set to **true**, defaults to **false** or **<none>**, preventing any **network-resources-injector** pod from running in the namespace. The recommended setting is **true**.

spec.enableOperatorWebhook

Specifies if any **operator-webhook** pods can run in the namespace. The **enableOperatorWebhook** field, if not specified in the CR or explicitly set to true, defaults to **false** or **<none>**, preventing any **operator-webhook** pod from running in the namespace. The recommended setting is **true**.

2. Apply the resource to your cluster by running the following command:

```
$ oc apply -f sriovOperatorConfig.yaml
```

10.2.2. SR-IOV Network Operator config custom resource

To customize the SR-IOV Network Operator, configure the **sriovoperatorconfig** custom resource. The reference lists the parameters available for controlling the global settings and deployment behavior of the Operator.

The following table describes the **sriovoperatorconfig** CR fields:

Table 10.1. SR-IOV Network Operator config custom resource

Field	Type	Description
metadata.name	**string**	Specifies the name of the SR-IOV Network Operator instance. The default value is **default**. Do not set a different value.
metadata.namespace	**string**	Specifies the namespace of the SR-IOV Network Operator instance. The default value is **openshift-sriov-network-operator**. Do not set a different value.
spec.configDaemonNodeSelector	**string**	Specifies the node selection to control scheduling the SR-IOV Network Config Daemon on selected nodes. By default, this field is not set and the Operator deploys the SR-IOV Network Config daemon set on compute nodes.
spec.disableDrain	**boolean**	Specifies whether to disable the node draining process or enable the node draining process when you apply a new policy to configure the NIC on a node. Setting this field to **true** facilitates software development and installing OpenShift Container Platform on a single node. By default, this field is not set. For single-node clusters, set this field to **true** after installing the Operator. This field must remain set to **true**.
spec.enableInjector	**boolean**	Specifies whether to enable or disable the Network Resources Injector daemon set.
spec.enableOperatorWebhook	**boolean**	Specifies whether to enable or disable the Operator Admission Controller webhook daemon set.
spec.logLevel	**integer**	Specifies the log verbosity level of the Operator. By default, this field is set to **0**, which shows only basic logs. Set to **2** to show all the available logs.
spec.featureGates	**map[string]bool**	Specifies whether to enable or disable the optional features. For example, **metricsExporter**.
spec.featureGates.metricsExporter	**boolean**	Specifies whether to enable or disable the SR-IOV Network Operator metrics. By default, this field is set to **false**.

Field	Type	Description
spec.featureGates.mellanoxFirmwareReset	**boolean**	Specifies whether to reset the firmware on virtual function (VF) changes in the SR-IOV Network Operator. Some chipsets, such as the Intel C740 Series, do not completely power off the PCI-E devices, which is required to configure VFs on NVIDIA/Mellanox NICs. By default, this field is set to **false**. IMPORTANT The **spec.featureGates.mellanoxFirmwareReset** parameter 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 Technology Preview Features Support Scope

10.2.3. About the Network Resources Injector

To automate network configuration for your workloads, use the Network Resources Injector. This Kubernetes Dynamic Admission Controller intercepts pod creation requests to automatically inject the necessary network resources and parameters defined for your cluster.

The Network Resources Injector provides the following capabilities:

- Mutation of resource requests and limits in a pod specification to add an SR-IOV resource name according to an SR-IOV network attachment definition annotation.
- Mutation of a pod specification with a Downward API volume to expose pod annotations, labels, and huge pages requests and limits. Containers that run in the pod can access the exposed information as files under the **/etc/podnetinfo** path.

The SR-IOV Network Operator enables the Network Resources Injector when the **enableInjector** is set to **true** in the **SriovOperatorConfig** CR. The **network-resources-injector** pod runs as a daemon set on all control plane nodes. The following is an example of Network Resources Injector pods running in a cluster with three control plane 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
```

By default, the **failurePolicy** field in the Network Resources Injector webhook is set to **Ignore**. This default setting prevents pod creation from being blocked if the webhook is unavailable.

If you set the **failurePolicy** field to **Fail**, and the Network Resources Injector webhook is unavailable, the webhook attempts to mutate all pod creation and update requests. This behavior can block pod creation and disrupt normal cluster operations. To prevent such issues, you can enable the **featureGates.resourceInjectorMatchCondition** feature in the **SriovOperatorConfig** object to limit the scope of the Network Resources Injector webhook. If this feature is enabled, the webhook applies only to pods with the secondary network annotation **k8s.v1.cni.cncf.io/networks**.

If you set the **failurePolicy** field to **Fail** after enabling the **resourceInjectorMatchCondition** feature, the webhook applies only to pods with the secondary network annotation **k8s.v1.cni.cncf.io/networks**. If the webhook is unavailable, the cluster still deploys pods without this annotation; this prevents unnecessary disruptions to cluster operations.

The **featureGates.resourceInjectorMatchCondition** feature is disabled by default. To enable this feature, set the **featureGates.resourceInjectorMatchCondition** field to **true** in the **SriovOperatorConfig** object.

Example SriovOperatorConfig object configuration

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovOperatorConfig
metadata:
  name: default
  namespace: sriov-network-operator
spec:
# ...
  featureGates:
    resourceInjectorMatchCondition: true
# ...
```

10.2.4. Disabling or enabling the Network Resources Injector

To control the automatic configuration of your cluster workloads, enable or disable the Network Resources Injector. By adjusting these settings you can better manage whether the Kubernetes Dynamic Admission Controller automatically injects network resources and parameters into pods during their creation.

Prerequisites

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

TIP

You can alternatively apply the following YAML to update the Operator:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovOperatorConfig
metadata:
  name: default
  namespace: openshift-sriov-network-operator
spec:
  enableInjector: <value>
# ...
```

10.2.5. About the SR-IOV Network Operator admission controller webhook

To validate network configurations and prevent errors, rely on the SR-IOV Network Operator admission controller webhook. This Kubernetes Dynamic Admission Controller intercepts API requests to ensure that your SR-IOV resource definitions and pod specifications comply with cluster policies.

The SR-IOV Network Operator Admission Controller webhook 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.

The SR-IOV Network Operator Admission Controller webhook is enabled by the Operator when the **enableOperatorWebhook** is set to **true** in the **SriovOperatorConfig** CR. The **operator-webhook** pod runs as a daemon set on all control plane nodes.

NOTE

Use caution when disabling the SR-IOV Network Operator Admission Controller webhook. You can disable the webhook under specific circumstances, such as troubleshooting, or if you want to use unsupported devices. For information about configuring unsupported devices, see "Configuring the SR-IOV Network Operator to use an unsupported NIC".

The following is an example of the Operator Admission Controller webhook pods running in a cluster with three control plane 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
```

Additional resources

- Configuring the SR-IOV Network Operator to use an unsupported NIC

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

To manage validation of your network configurations, enable or disable the SR-IOV Network Operator admission controller webhook. When enabled, the Operator automatically verifies SR-IOV resource definitions and pod specifications against cluster policies.

Prerequisites

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

TIP

You can alternatively apply the following YAML to update the Operator:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovOperatorConfig
metadata:
  name: default
  namespace: openshift-sriov-network-operator
spec:
  enableOperatorWebhook: <value>
# ...
```

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

To specify which nodes host the SR-IOV Network Config daemon, configure a custom node selector by using node labels. By completing this task, you can restrict deployment to specific nodes instead of the default compute nodes.

The SR-IOV Network Config daemon discovers and configures the SR-IOV network devices on cluster nodes. By default, the daemon is deployed to all the compute nodes in the cluster.

IMPORTANT

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

Procedure

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

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

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

TIP

You can alternatively apply the following YAML to update the Operator:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovOperatorConfig
metadata:
  name: default
  namespace: openshift-sriov-network-operator
spec:
  configDaemonNodeSelector:
    <node_label>
# ...
```

10.2.8. Configuring the SR-IOV Network Operator for single node installations

By default, the SR-IOV Network Operator drains workloads from a node before every policy change. The Operator performs this action to ensure that no workloads are using the virtual functions before the reconfiguration. As a result, you must configure the Operator to not drain workloads from the single node.

For installations on a single node, other nodes do not receive the workloads.

IMPORTANT

After performing the following procedure to disable draining workloads, you must remove any workload that uses an SR-IOV network interface before you change any SR-IOV network node policy.

Prerequisites

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

- Log in as a user with **cluster-admin** privileges.
- You must have installed the SR-IOV Network Operator.

Procedure

- To set the **disableDrain** field to **true** and the **configDaemonNodeSelector** field to **node-role.kubernetes.io/master: ""**, enter the following command:

```
$ oc patch sriovoperatorconfig default --type=merge -n openshift-sriov-network-operator --
patch '{ "spec": { "disableDrain": true, "configDaemonNodeSelector": { "node-
role.kubernetes.io/master": "" } } }'
```

TIP

You can alternatively apply the following YAML to update the Operator:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovOperatorConfig
metadata:
  name: default
  namespace: openshift-sriov-network-operator
spec:
  disableDrain: true
  configDaemonNodeSelector:
   node-role.kubernetes.io/master: ""
# ...
```

10.2.8.1. Deploying the SR-IOV Operator for hosted control planes

After you configure and deploy your hosting service cluster, you can create a subscription to the SR-IOV Operator on a hosted cluster. The SR-IOV pod runs on worker machines rather than the control plane.

Prerequisites

You must configure and deploy the hosted cluster on AWS.

Procedure

1. Create a namespace and an Operator group:

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

2. Create a subscription to the SR-IOV Operator:

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: sriov-network-operator-subsription
  namespace: openshift-sriov-network-operator
spec:
  channel: stable
  name: sriov-network-operator
  config:
    nodeSelector:
      node-role.kubernetes.io/worker: ""
  source: redhat-operators
  sourceNamespace: openshift-marketplace
```

Verification

1. To verify that the SR-IOV Operator is ready, run the following command and view the resulting output:

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

Example output

```
NAME                                         DISPLAY                   VERSION               REPLACES
PHASE
sriov-network-operator.4.18.0-202211021237   SR-IOV Network Operator   4.18.0-
202211021237   sriov-network-operator.4.18.0-202210290517   Succeeded
```

2. To verify that the SR-IOV pods are deployed, run the following command:

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

10.2.9. About the SR-IOV network metrics exporter

To monitor the networking activity of SR-IOV pods, enable the SR-IOV network metrics exporter. This tool exposes metrics for SR-IOV virtual functions (VFs) in Prometheus format, so that you can query and visualize data through the OpenShift Container Platform web console.

When you query the SR-IOV VF metrics by using the web console, the SR-IOV network metrics exporter fetches and returns the VF network statistics along with the name and namespace of the pod that the VF is attached to.

The following table describes the SR-IOV VF metrics that the metrics exporter reads and exposes in Prometheus format:

Table 10.2. SR-IOV VF metrics

Metric	Description	Example PromQL query to examine the VF metric

Metric	Description	Example PromQL query to examine the VF metric
sriov_vf_rx_bytes	Received bytes per virtual function.	**sriov_vf_rx_bytes * on (pciAddr,node) group_left(pod,namespace,dev_type) sriov_kubepoddevice**
sriov_vf_tx_bytes	Transmitted bytes per virtual function.	**sriov_vf_tx_bytes * on (pciAddr,node) group_left(pod,namespace,dev_type) sriov_kubepoddevice**
sriov_vf_rx_packets	Received packets per virtual function.	**sriov_vf_rx_packets * on (pciAddr,node) group_left(pod,namespace,dev_type) sriov_kubepoddevice**
sriov_vf_tx_packets	Transmitted packets per virtual function.	**sriov_vf_tx_packets * on (pciAddr,node) group_left(pod,namespace,dev_type) sriov_kubepoddevice**
sriov_vf_rx_dropped	Dropped packets upon receipt per virtual function.	**sriov_vf_rx_dropped * on (pciAddr,node) group_left(pod,namespace,dev_type) sriov_kubepoddevice**
sriov_vf_tx_dropped	Dropped packets during transmission per virtual function.	**sriov_vf_tx_dropped * on (pciAddr,node) group_left(pod,namespace,dev_type) sriov_kubepoddevice**
sriov_vf_rx_multicast	Received multicast packets per virtual function.	**sriov_vf_rx_multicast * on (pciAddr,node) group_left(pod,namespace,dev_type) sriov_kubepoddevice**

Metric	Description	Example PromQL query to examine the VF metric
sriov_vf_rx_broadcast	Received broadcast packets per virtual function.	**sriov_vf_rx_broadcast * on (pciAddr,node) group_left(pod,namespace,dev_type) sriov_kubepoddevice**
sriov_kubepoddevice	Virtual functions linked to active pods.	-

You can also combine these queries by using the **kube-state-metrics** tool to get more information about the SR-IOV pods. For example, you can use the following query to get the VF network statistics along with the application name from the standard Kubernetes pod label:

```
(sriov_vf_tx_packets * on (pciAddr,node)  group_left(pod,namespace)  sriov_kubepoddevice) * on
(pod,namespace) group_left (label_app_kubernetes_io_name) kube_pod_labels
```

10.2.9.1. Enabling the SR-IOV network metrics exporter

To enable the SR-IOV network metrics exporter, set the **spec.featureGates.metricsExporter** field to **true**. Because the exporter is disabled by default, you must explicitly activate this feature gate to start exposing metrics for your SR-IOV devices.

IMPORTANT

When the metrics exporter is enabled, the SR-IOV Network Operator deploys the metrics exporter only on nodes with SR-IOV capabilities.

Prerequisites

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

Procedure

1. Enable cluster monitoring by running the following command:

```
$ oc label ns/openshift-sriov-network-operator openshift.io/cluster-monitoring=true
```

   To enable cluster monitoring, you must add the **openshift.io/cluster-monitoring=true** label in the namespace where you have installed the SR-IOV Network Operator.

2. Set the **spec.featureGates.metricsExporter** field to **true** by running the following command:

```
$ oc patch -n openshift-sriov-network-operator sriovoperatorconfig/default \
    --type='merge' -p='{"spec": {"featureGates": {"metricsExporter": true}}}'
```

Verification

1. Check that the SR-IOV network metrics exporter is enabled by running the following command:

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

Example output

```
NAME                                  READY   STATUS    RESTARTS   AGE
operator-webhook-hzfg4                1/1     Running   0          5d22h
sriov-network-config-daemon-tr54m     1/1     Running   0          5d22h
sriov-network-metrics-exporter-z5d7t  1/1     Running   0          10s
sriov-network-operator-cc6fd88bc-9bsmt  1/1   Running   0          5d22h
```

Ensure that **sriov-network-metrics-exporter** pod is in the **READY** state.

2. Optional: Examine the SR-IOV virtual function (VF) metrics by using the OpenShift Container Platform web console. For more information, see "Querying metrics".

Additional resources

- Querying metrics for all projects with the monitoring dashboard
- Querying metrics for user-defined projects as a developer
- Configuring an SR-IOV network device
- Uninstalling the SR-IOV Network Operator

10.3. UNINSTALLING THE SR-IOV NETWORK OPERATOR

To uninstall the SR-IOV Network Operator, you must delete any running SR-IOV workloads, uninstall the Operator, and delete the webhooks that the Operator used.

10.3.1. Uninstalling the SR-IOV Network Operator

You can remove the SR-IOV Network Operator from your cluster by uninstalling the Operator. This ensures that the Operator and its associated resources are deleted when you no longer need to manage SR-IOV network devices.

Prerequisites

- You have access to an OpenShift Container Platform cluster using an account with **cluster-admin** permissions.
- You have the SR-IOV Network Operator installed.

Procedure

1. Delete all SR-IOV custom resources (CRs):

```
$ oc delete sriovnetwork -n openshift-sriov-network-operator --all
```

```
$ oc delete sriovnetworknodepolicy -n openshift-sriov-network-operator --all
```

```
$ oc delete sriovibnetwork -n openshift-sriov-network-operator --all
```

```
$ oc delete sriovoperatorconfigs -n openshift-sriov-network-operator --all
```

2. Follow the instructions in the "Deleting Operators from a cluster" section to remove the SR-IOV Network Operator from your cluster.

3. Delete the SR-IOV custom resource definitions that remain in the cluster after the SR-IOV Network Operator is uninstalled:

```
$ oc delete crd sriovibnetworks.sriovnetwork.openshift.io
```

```
$ oc delete crd sriovnetworknodepolicies.sriovnetwork.openshift.io
```

```
$ oc delete crd sriovnetworknodestates.sriovnetwork.openshift.io
```

```
$ oc delete crd sriovnetworkpoolconfigs.sriovnetwork.openshift.io
```

```
$ oc delete crd sriovnetworks.sriovnetwork.openshift.io
```

```
$ oc delete crd sriovoperatorconfigs.sriovnetwork.openshift.io
```

4. Delete the SR-IOV Network Operator namespace:

```
$ oc delete namespace openshift-sriov-network-operator
```

10.3.2. Additional resources

- Deleting Operators from a cluster