OpenShift Container Platform 4.18

Ingress and load balancing

Exposing services and managing external traffic in OpenShift Container Platform

Last Updated: 2026-04-01

OpenShift Container Platform 4.18 Ingress and load balancing

Exposing services and managing external traffic in OpenShift Container Platform

Legal Notice

Abstract

This document explains how to configure routes, manage ingress traffic, and implement various load balancing solutions in OpenShift Container Platform.

Table of Contents

CHAPTER 1. ROUTES

1.1. CREATING BASIC ROUTES

If you have unencrypted HTTP, you can create a basic route with a route object.

1.1.1. Creating an HTTP-based route

You can use the following procedure to create a simple HTTP-based route to a web application, using the **hello-openshift** application as an example.

You can create a route to host your application at a public URL. The route can either be secure or unsecured, depending on the network security configuration of your application. An HTTP-based route is an unsecured route that uses the basic HTTP routing protocol and exposes a service on an unsecured application port.

Prerequisites

- You installed the OpenShift CLI (**oc**).
- You are logged in as an administrator.
- You have a web application that exposes a port and a TCP endpoint listening for traffic on the port.

Procedure

1. Create a project called **hello-openshift** by running the following command:

```
$ oc new-project hello-openshift
```

2. Create a pod in the project by running the following command:

```
$ oc create -f https://raw.githubusercontent.com/openshift/origin/master/examples/hello-openshift/hello-pod.json
```

3. Create a service called **hello-openshift** by running the following command:

```
$ oc expose pod/hello-openshift
```

4. Create an unsecured route to the **hello-openshift** application by running the following command:

```
$ oc expose svc hello-openshift
```

Verification

- To verify that the **route** resource that you created, run the following command:

```
$ oc get routes -o yaml hello-openshift
```

Example YAML definition of the created unsecured route

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: hello-openshift
spec:
  host: www.example.com
  port:
    targetPort: 8080
  to:
    kind: Service
    name: hello-openshift
```

where:

host

Specifies an alias DNS record that points to the service. This field can be any valid DNS name, such as **www.example.com**. The DNS name must follow DNS952 subdomain conventions. If not specified, a route name is automatically generated.

targetPort

Specifies the target port on pods that is selected by the service that this route points to.

NOTE

To display your default ingress domain, run the following command:

```
$ oc get ingresses.config/cluster -o jsonpath={.spec.domain}
```

1.1.2. Path-based routes

To serve multiple applications by using a single hostname, configure path-based routes. This HTTP-based configuration directs traffic to specific services by comparing the URL path component, ensuring requests match the most specific route defined.

The following table shows example routes and their accessibility:

Table 1.1. Route availability

Route	When compared to	Accessible
www.example.com/test	*www.example.com/test*	Yes
	www.example.com	No
www.example.com/test and *www.example.com*	*www.example.com/test*	Yes
	www.example.com	Yes

Route	When compared to	Accessible
www.example.com	*www.example.com/text*	Yes (Matched by the host, not the route)
	www.example.com	Yes

Example of an unsecured route with a path

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: route-unsecured
spec:
  host: www.example.com
  path: "/test"
  to:
    kind: Service
    name: service-name
```

- **spec.host**: Specifies the path attribute for a path-based route.

NOTE

Path-based routing is not available when using passthrough TLS, as the router does not terminate TLS in that case and cannot read the contents of the request.

1.1.3. Creating a route for Ingress Controller sharding

To host applications at specific URLs and balance traffic load in OpenShift Container Platform, configure Ingress Controller sharding. By sharding, you can isolate traffic for specific workloads or tenants, ensuring efficient resource management across your cluster.

The following procedure describes how to create a route for Ingress Controller sharding, using the **hello-openshift** application as an example.

Prerequisites

- You installed the OpenShift CLI (**oc**).
- You are logged in as a project administrator.
- You have a web application that exposes a port and an HTTP or TLS endpoint listening for traffic on the port.
- You have configured the Ingress Controller for sharding.

Procedure

1. Create a project called **hello-openshift** by running the following command:

```
$ oc new-project hello-openshift
```

2. Create a pod in the project by running the following command:

```
$ oc create -f https://raw.githubusercontent.com/openshift/origin/master/examples/hello-openshift/hello-pod.json
```

3. Create a service called **hello-openshift** by running the following command:

```
$ oc expose pod/hello-openshift
```

4. Create a route definition called **hello-openshift-route.yaml**:

YAML definition of the created route for sharding

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  labels:
    type: sharded
  name: hello-openshift-edge
  namespace: hello-openshift
spec:
  subdomain: hello-openshift
  tls:
    termination: edge
  to:
    kind: Service
    name: hello-openshift
```

where:

type

Specifies both the label key and its corresponding label value must match the ones specified in the Ingress Controller. In this example, the Ingress Controller has the label key and value **type: sharded**.

subdomain

Specifies the route gets exposed by using the value of the **subdomain** field. When you specify the **subdomain** field, you must leave the hostname unset. If you specify both the **host** and **subdomain** fields, then the route uses the value of the **host** field, and ignore the **subdomain** field.

5. Use **hello-openshift-route.yaml** to create a route to the **hello-openshift** application by running the following command:

```
$ oc -n hello-openshift create -f hello-openshift-route.yaml
```

Verification

- Get the status of the route with the following command:

```
$ oc -n hello-openshift get routes/hello-openshift-edge -o yaml
```

The resulting **Route** resource should look similar to the following:

Example output

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  labels:
    type: sharded
  name: hello-openshift-edge
  namespace: hello-openshift
spec:
  subdomain: hello-openshift
  tls:
    termination: edge
  to:
    kind: Service
    name: hello-openshift
status:
  ingress:
  - host: hello-openshift.<apps-sharded.basedomain.example.net>
    routerCanonicalHostname: router-sharded.<apps-sharded.basedomain.example.net>
    routerName: sharded
```

where:

host

Specifies the hostname the Ingress Controller, or router, uses to expose the route. The value of the **host** field is automatically determined by the Ingress Controller, and uses its domain. In this example, the domain of the Ingress Controller is **<apps-sharded.basedomain.example.net>**.

<apps-sharded.basedomain.example.net>

Specifies the hostname of the Ingress Controller. If the hostname is not set, the route can use a subdomain instead. When you specify a subdomain, you automatically use the domain of the Ingress Controller that exposes the route. When a route is exposed by multiple Ingress Controllers, the route is hosted at multiple URLs.

routerName

Specifies the name of the Ingress Controller. In this example, the Ingress Controller has the name **sharded**.

1.1.4. Creating a route through an Ingress object

To integrate ecosystem components that require Ingress resources, configure an Ingress object. OpenShift Container Platform automatically manages the lifecycle of the corresponding route objects, creating and deleting them to ensure seamless connectivity.

Procedure

1. Define an Ingress object in the OpenShift Container Platform console or by entering the **oc create** command:

YAML Definition of an Ingress

-

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: frontend
  annotations:
    route.openshift.io/termination: "reencrypt"
    route.openshift.io/destination-ca-certificate-secret: secret-ca-cert
spec:
  rules:
  - host: www.example.com
    http:
      paths:
      - backend:
          service:
            name: frontend
            port:
              number: 443
        path: /
        pathType: Prefix
  tls:
  - hosts:
    - www.example.com
    secretName: example-com-tls-certificate
# ...
```

where:

route.openshift.io/termination

Specifies the **route.openshift.io/termination** annotation. You can configure the **spec.tls.termination** parameter of the **Route** because **Ingress** does not have this parameter. The accepted values are **edge**, **passthrough**, and **reencrypt**. All other values are silently ignored. When the annotation value is unset, **edge** is the default route. The TLS certificate details must be defined in the template file to implement the default edge route.

rules.host

Specifies an explicit hostname for the **Ingress** object. Mandatory parameter. You can use the **<host_name>.<cluster_ingress_domain>** syntax, for example **apps.openshiftdemos.com**, to take advantage of the ***.<cluster_ingress_domain>** wildcard DNS record and serving certificate for the cluster. Otherwise, you must ensure that there is a DNS record for the chosen hostname.

destination-ca-certificate-secret

Specifies the **route.openshift.io/destination-ca-certificate-secret** annotation. The annotation can be used on an Ingress object to define a route with a custom destination certificate (CA). The annotation references a kubernetes secret, **secret-ca-cert** that will be inserted into the generated route.

a. If you specify the **passthrough** value in the **route.openshift.io/termination** annotation, set **path** to **''** and **pathType** to **ImplementationSpecific** in the spec:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
# ...
  spec:
    rules:
```

```
    - host: www.example.com
      http:
        paths:
        - path: ''
          pathType: ImplementationSpecific
          backend:
            service:
              name: frontend
              port:
                number: 443
# ...
```

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

b. To specify a route object with a destination CA from an ingress object, you must create a **kubernetes.io/tls** or **Opaque** type secret with a certificate in PEM-encoded format in the **data.tls.crt** specifier of the secret.

2. List your routes:

```
$ oc get routes
```

The result includes an autogenerated route whose name starts with **frontend-**:

```
NAME             HOST/PORT         PATH   SERVICES   PORT   TERMINATION
WILDCARD
frontend-gnztq   www.example.com          frontend   443    reencrypt/Redirect   None
```

YAML definition example of an autogenerated route

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: frontend-gnztq
  ownerReferences:
  - apiVersion: networking.k8s.io/v1
    controller: true
    kind: Ingress
    name: frontend
    uid: 4e6c59cc-704d-4f44-b390-617d879033b6
spec:
  host: www.example.com
  path: /
  port:
    targetPort: https
  tls:
    certificate: |
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
    insecureEdgeTerminationPolicy: Redirect
    key: |
      -----BEGIN RSA PRIVATE KEY-----
```

```
    [...]
    -----END RSA PRIVATE KEY-----
  termination: reencrypt
  destinationCACertificate: |
    -----BEGIN CERTIFICATE-----
    [...]
    -----END CERTIFICATE-----
to:
  kind: Service
  name: frontend
```

1.2. SECURING ROUTES

You can secure a route with HTTP strict transport security (HSTS).

1.2.1. HTTP Strict Transport Security

To enhance security and optimize website performance, use the HTTP Strict Transport Security (HSTS) policy. This mechanism signals browsers to use only HTTPS traffic on the route host, eliminating the need for HTTP redirects and speeding up user interactions.

When HSTS policy is enforced, HSTS adds a Strict Transport Security header to HTTP and HTTPS responses from the site. You can use the **insecureEdgeTerminationPolicy** value in a route to redirect HTTP to HTTPS. When HSTS is enforced, the client changes all requests from the HTTP URL to HTTPS before the request is sent, eliminating the need for a redirect.

Cluster administrators can configure HSTS to do the following:

- Enable HSTS per-route
- Disable HSTS per-route
- Enforce HSTS per-domain, for a set of domains, or use namespace labels in combination with domains

IMPORTANT

HSTS works only with secure routes, either edge-terminated or re-encrypt. The configuration is ineffective on HTTP or passthrough routes.

1.2.1.1. Enabling HTTP Strict Transport Security per-route

To enforce secure HTTPS connections for specific applications, enable HTTP Strict Transport Security (HSTS) on a per-route basis. Applying the **haproxy.router.openshift.io/hsts_header** annotation to edge and re-encrypt routes ensures that browsers reject unencrypted traffic.

Prerequisites

- You are logged in to the cluster with a user with administrator privileges for the project.
- You installed the OpenShift CLI (**oc**).

Procedure

- To enable HSTS on a route, add the **haproxy.router.openshift.io/hsts_header** value to the edge-terminated or re-encrypt route. You can use the **oc annotate** tool to do this by running the following command. To properly run the command, ensure that the semicolon (**;**) in the **haproxy.router.openshift.io/hsts_header** route annotation is also surrounded by double quotation marks (**""**).

  Example annotate command that sets the maximum age to 31536000 ms (approximately 8.5 hours)

  ```
  $ oc annotate route <route_name> -n <namespace> --overwrite=true
  "haproxy.router.openshift.io/hsts_header=max-age=31536000;\
  includeSubDomains;preload"
  ```

  Example route configured with an annotation

  ```
  apiVersion: route.openshift.io/v1
  kind: Route
  metadata:
    annotations:
      haproxy.router.openshift.io/hsts_header: max-age=31536000;includeSubDomains;preload
  # ...
  spec:
    host: def.abc.com
    tls:
      termination: "reencrypt"
      ...
    wildcardPolicy: "Subdomain"
  # ...
  ```

  where:

  max-age

  Specifies the measurement of the length of time, in seconds, for the HSTS policy. If set to **0**, it negates the policy.

  includeSubDomains

  Specifies that all subdomains of the host must have the same HSTS policy as the host. Optional parameter.

  preload

  Specifies that the site is included in the HSTS preload list when **max-age** is greater than **0**. For example, sites such as Google can construct a list of sites that have **preload** set. Browsers can then use these lists to determine which sites they can communicate with over HTTPS, even before they have interacted with the site. Without **preload** set, browsers must have interacted with the site over HTTPS, at least once, to get the header. Optional parameter.

1.2.1.2. Disabling HTTP Strict Transport Security per-route

To allow unencrypted connections or troubleshoot access issues, disable HTTP Strict Transport Security (HSTS) for a specific route. Setting the **max-age** route annotation to **0** instructs browsers to stop enforcing HTTPS requirements on the route host.

Prerequisites

- You are logged in to the cluster with a user with administrator privileges for the project.
- You installed the OpenShift CLI (**oc**).

Procedure

- To disable HSTS, enter the following to set the **max-age** value in the route annotation to **0**:

```
$ oc annotate route <route_name> -n <namespace> --overwrite=true
"haproxy.router.openshift.io/hsts_header"="max-age=0"
```

TIP

You can alternatively apply the following YAML to create the config map for disabling HSTS per-route:

```
kind: Route
apiVersion: route.openshift.io/v1
metadata:
  annotations:
    haproxy.router.openshift.io/hsts_header: max-age=0
```

- To disable HSTS for every route in a namespace, enter the following command:

```
$ oc annotate route --all -n <namespace> --overwrite=true
"haproxy.router.openshift.io/hsts_header"="max-age=0"
```

Verification

1. To query the annotation for all routes, enter the following command:

```
$ oc get route  --all-namespaces -o go-template='{{range .items}}{{if .metadata.annotations}}
{{$a := index .metadata.annotations "haproxy.router.openshift.io/hsts_header"}}{{$n :=
.metadata.name}}{{with $a}}Name: {{$n}} HSTS: {{$a}}{{"\n"}}{{else}}{{""}}{{end}}{{end}}
{{end}}'
```

Example output

```
Name: routename HSTS: max-age=0
```

1.2.1.3. Enforcing HTTP Strict Transport Security per-domain

To enforce HTTP Strict Transport Security (HSTS) per-domain for secure routes, add a **requiredHSTSPolicies** record to the Ingress spec to capture the configuration of the HSTS policy.

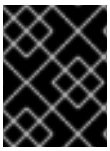
If you configure a **requiredHSTSPolicy** to enforce HSTS, then any newly created route must be configured with a compliant HSTS policy annotation.

NOTE

To handle upgraded clusters with non-compliant HSTS routes, you can update the manifests at the source and apply the updates.

NOTE

You cannot use **oc expose route** or **oc create route** commands to add a route in a domain that enforces HSTS, because the API for these commands does not accept annotations.



IMPORTANT

HSTS cannot be applied to insecure, or non-TLS routes, even if HSTS is requested for all routes globally.

Prerequisites

- You are logged in to the cluster with a user with administrator privileges for the project.
- You installed the OpenShift CLI (**oc**).

Procedure

1. Edit the Ingress configuration YAML by running the following command and updating fields as needed:

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

Example HSTS policy

```
apiVersion: config.openshift.io/v1
kind: Ingress
metadata:
  name: cluster
spec:
  domain: 'hello-openshift-default.apps.username.devcluster.openshift.com'
  requiredHSTSPolicies: 1
  - domainPatterns: 2
    - '*hello-openshift-default.apps.username.devcluster.openshift.com'
    - '*hello-openshift-default2.apps.username.devcluster.openshift.com'
    namespaceSelector: 3
      matchLabels:
        myPolicy: strict
    maxAge: 4
      smallestMaxAge: 1
      largestMaxAge: 31536000
    preloadPolicy: RequirePreload 5
    includeSubDomainsPolicy: RequireIncludeSubDomains 6
  - domainPatterns:
    - 'abc.example.com'
    - '*xyz.example.com'
    namespaceSelector:
      matchLabels: {}
```

```
maxAge: {}
preloadPolicy: NoOpinion
includeSubDomainsPolicy: RequireNoIncludeSubDomains
```

1. Required. **requiredHSTSPolicies** are validated in order, and the first matching **domainPatterns** applies.
2. Required. You must specify at least one **domainPatterns** hostname. Any number of domains can be listed. You can include multiple sections of enforcing options for different **domainPatterns**.
3. Optional. If you include **namespaceSelector**, it must match the labels of the project where the routes reside, to enforce the set HSTS policy on the routes. Routes that only match the **namespaceSelector** and not the **domainPatterns** are not validated.
4. Required. **max-age** measures the length of time, in seconds, that the HSTS policy is in effect. This policy setting allows for a smallest and largest **max-age** to be enforced.
   - The **largestMaxAge** value must be between **0** and **2147483647**. It can be left unspecified, which means no upper limit is enforced.
   - The **smallestMaxAge** value must be between **0** and **2147483647**. Enter **0** to disable HSTS for troubleshooting, otherwise enter **1** if you never want HSTS to be disabled. It can be left unspecified, which means no lower limit is enforced.
5. Optional. Including **preload** in **haproxy.router.openshift.io/hsts_header** allows external services to include this site in their HSTS preload lists. Browsers can then use these lists to determine which sites they can communicate with over HTTPS, before they have interacted with the site. Without **preload** set, browsers need to interact at least once with the site to get the header. **preload** can be set with one of the following:
   - **RequirePreload**: **preload** is required by the **RequiredHSTSPolicy**.
   - **RequireNoPreload**: **preload** is forbidden by the **RequiredHSTSPolicy**.
   - **NoOpinion**: **preload** does not matter to the **RequiredHSTSPolicy**.
6. Optional. **includeSubDomainsPolicy** can be set with one of the following:
   - **RequireIncludeSubDomains**: **includeSubDomains** is required by the **RequiredHSTSPolicy**.
   - **RequireNoIncludeSubDomains**: **includeSubDomains** is forbidden by the **RequiredHSTSPolicy**.
   - **NoOpinion**: **includeSubDomains** does not matter to the **RequiredHSTSPolicy**.

2. You can apply HSTS to all routes in the cluster or in a particular namespace by entering the **oc annotate command**.
   - To apply HSTS to all routes in the cluster, enter the **oc annotate command**. For example:

```
$ oc annotate route --all --all-namespaces --overwrite=true
"haproxy.router.openshift.io/hsts_header"="max-age=31536000"
```

- To apply HSTS to all routes in a particular namespace, enter the **oc annotate command**. For example:

```
$ oc annotate route --all -n my-namespace --overwrite=true
"haproxy.router.openshift.io/hsts_header"="max-age=31536000"
```

Verification

You can review the HSTS policy you configured. For example:

- To review the **maxAge** set for required HSTS policies, enter the following command:

```
$ oc get clusteroperator/ingress -n openshift-ingress-operator -o jsonpath='{range
.spec.requiredHSTSPolicies[*]}{.spec.requiredHSTSPolicies.maxAgePolicy.largestMaxAge}
{"\n"}{end}'
```

- To review the HSTS annotations on all routes, enter the following command:

```
$ oc get route  --all-namespaces -o go-template='{{range .items}}{{if .metadata.annotations}}
{{$a := index .metadata.annotations "haproxy.router.openshift.io/hsts_header"}}{{$n :=
.metadata.name}}{{with $a}}Name: {{$n}} HSTS: {{$a}}{{"\n"}}{{else}}{{""}}{{end}}{{end}}
{{end}}'
```

Example output

```
Name: <_routename_> HSTS: max-age=31536000;preload;includeSubDomains
```

1.3. CONFIGURING ROUTES

To customise route configuration for specific traffic behaviors, apply annotations, headers, and cookies. By using these mechanisms, you can define granular routing rules, extending standard capabilities to meet complex application requirements.

1.3.1. Configuring route timeouts

You can configure the default timeouts for an existing route when you have services in need of a low timeout, which is required for Service Level Availability (SLA) purposes, or a high timeout, for cases with a slow back end.

IMPORTANT

If you configured a user-managed external load balancer in front of your OpenShift Container Platform cluster, ensure that the timeout value for the user-managed external load balancer is higher than the timeout value for the route. This configuration prevents network congestion issues over the network that your cluster uses.

Prerequisites

- You deployed an Ingress Controller on a running cluster.

Procedure

1. Using the **oc annotate** command, add the timeout to the route:

```
$ oc annotate route <route_name> \
    --overwrite haproxy.router.openshift.io/timeout=<timeout><time_unit>
```

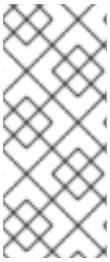
- **<timeout>**: Supported time units are microseconds (us), milliseconds (ms), seconds (s), minutes (m), hours (h), or days (d).
  The following example sets a timeout of two seconds on a route named **myroute**:

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

1.3.2. 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 1.2. 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

1.3.3. Setting or deleting HTTP request and response headers in a route

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 enable a web application to serve content in alternate locations for specific routes if that content is written in multiple languages, even if there is a default global location specified by the Ingress Controller serving the routes.

The following procedure creates a route that sets the Content-Location HTTP request header so that the URL associated with the application, **https://app.example.com**, directs to the location **https://app.example.com/lang/en-us**. Directing application traffic to this location means that anyone using that specific route is accessing web content written in American English.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You are logged into an OpenShift Container Platform cluster as a project administrator.
- You have a web application that exposes a port and an HTTP or TLS endpoint listening for traffic on the port.

Procedure

1. Create a route definition and save it in a file called **app-example-route.yaml**:

YAML definition of the created route with HTTP header directives

```
apiVersion: route.openshift.io/v1
kind: Route
# ...
spec:
  host: app.example.com
  tls:
    termination: edge
  to:
    kind: Service
    name: app-example
  httpHeaders:
    actions:
      response:
      - name: Content-Location
        action:
          type: Set
          set:
            value: /lang/en-us
# ...
```

where:

actions

Specifies the list of actions you want to perform on the HTTP headers.

response

Specifies the type of header you want to change. In this case, a response header.

response.name

Specifies the name of the header you want to change. For a list of available headers you can set or delete, see *HTTP header configuration*.

action.type

Specifies the type of action being taken on the header. This field can have the value **Set** or **Delete**.

set.value

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, the value is set to the relative location of the content.

2. Create a route to your existing web application using the newly created route definition:

```
$ oc -n app-example create -f app-example-route.yaml
```

For HTTP request headers, the actions specified in the route definitions are executed after any actions performed on HTTP request headers in the Ingress Controller. This means that any values set for those request headers in a route will take precedence over the ones set in the Ingress Controller. For more information on the processing order of HTTP headers, see *HTTP header configuration*.

1.3.4. Using cookies to keep route statefulness

To maintain stateful application traffic during pod restarts or scaling events, configure sticky sessions by using cookies. By using this method, you ensure that all incoming traffic reaches the same endpoint, preventing state loss even if the specific endpoint pod changes.

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

NOTE

Cookies cannot be set on passthrough routes, because the HTTP traffic cannot be seen. Instead, a number is calculated based on the source IP address, which determines the backend.

If backends change, the traffic can be directed to the wrong server, making it less sticky. If you are using a load balancer, which hides source IP, the same number is set for all connections and traffic is sent to the same pod.

1.3.4.1. Annotating a route with a cookie

To enable applications to manage session persistence and load distribution, annotate the route with a custom cookie name. Overwriting the default cookie allows the backend application to identify and delete the specific cookie, forcing endpoint re-selection when necessary.

When a server is overloaded, the server tries to remove the requests from the client and redistribute the requests to other endpoints.

Procedure

1. Annotate the route with the specified cookie name:

```
$ oc annotate route <route_name> router.openshift.io/cookie_name="<cookie_name>"
```

where:

<route_name>
Specifies the name of the route.

<cookie_name>
Specifies the name for the cookie.
For example, to annotate the route **my_route** with the cookie name **my_cookie**:

```
$ oc annotate route my_route router.openshift.io/cookie_name="my_cookie"
```

2. Capture the route hostname in a variable:

```
$ ROUTE_NAME=$(oc get route <route_name> -o jsonpath='{.spec.host}')
```

where:

<route_name>

Specifies the name of the route.

3. Save the cookie, and then access the route:

```
$ curl $ROUTE_NAME -k -c /tmp/cookie_jar
```

Use the cookie saved by the previous command when connecting to the route:

```
$ curl $ROUTE_NAME -k -b /tmp/cookie_jar
```

1.3.5. Route-specific annotations

The Ingress Controller can set the default options for all the routes it exposes. An individual route can override some of these defaults by providing specific configurations in its annotations. Red Hat does not support adding a route annotation to an operator-managed route.

IMPORTANT

To create an allow list with multiple source IPs or subnets, use a space-delimited list. Any other delimiter type causes the list to be ignored without a warning or error message.

Table 1.3. Route annotations

Variable	Description
haproxy.router.openshift.io/balance	Sets the load-balancing algorithm. Available options are **random**, **source**, **roundrobin**[1], and **leastconn**. The default value is **source** for TLS passthrough routes. For all other routes, the default is **random**.
haproxy.router.openshift.io/disable_cookies	Disables the use of cookies to track related connections. If set to **'true'** or **'TRUE'**, the balance algorithm is used to choose which back-end serves connections for each incoming HTTP request.
router.openshift.io/cookie_name	Specifies an optional cookie to use for this route. The name must consist of any combination of upper and lower case letters, digits, "_", and "-". The default is the hashed internal key name for the route.
haproxy.router.openshift.io/pod-concurrent-connections	Sets the maximum number of connections that are allowed to a backing pod from a router. Note: If there are multiple pods, each can have this many connections. If you have multiple routers, there is no coordination among them, each may connect this many times. If not set, or set to 0, there is no limit.

Variable	Description
haproxy.router.openshift.io/rate-limit-connections	Setting **'true'** or **'TRUE'** enables rate limiting functionality which is implemented through stick-tables on the specific backend per route. Note: Using this annotation provides basic protection against denial-of-service attacks.
haproxy.router.openshift.io/rate-limit-connections.concurrent-tcp	Limits the number of concurrent TCP connections made through the same source IP address. It accepts a numeric value. Note: Using this annotation provides basic protection against denial-of-service attacks.
haproxy.router.openshift.io/rate-limit-connections.rate-http	Limits the rate at which a client with the same source IP address can make HTTP requests. It accepts a numeric value. Note: Using this annotation provides basic protection against denial-of-service attacks.
haproxy.router.openshift.io/rate-limit-connections.rate-tcp	Limits the rate at which a client with the same source IP address can make TCP connections. It accepts a numeric value.
router.openshift.io/haproxy.health.check.interval	Sets the interval for the back-end health checks. (TimeUnits)
haproxy.router.openshift.io/ip_allowlist	Sets an allowlist for the route. The allowlist is a space-separated list of IP addresses and CIDR ranges for the approved source addresses. Requests from IP addresses that are not in the allowlist are dropped. The maximum number of IP addresses and CIDR ranges directly visible in the **haproxy.config** file is 61. [2]
haproxy.router.openshift.io/hsts_header	Sets a Strict-Transport-Security header for the edge terminated or re-encrypt route.
haproxy.router.openshift.io/rewrite-target	Sets the rewrite path of the request on the backend.

Variable	Description
router.openshift.io/cookie-same-site	Sets a value to restrict cookies. The values are: **Lax**: the browser does not send cookies on cross-site requests, but does send cookies when users navigate to the origin site from an external site. This is the default browser behavior when the **SameSite** value is not specified. **Strict**: the browser sends cookies only for same-site requests. **None**: the browser sends cookies for both cross-site and same-site requests. This value is applicable to re-encrypt and edge routes only. For more information, see the SameSite cookies documentation.
haproxy.router.openshift.io/set-forwarded-headers	Sets the policy for handling the **Forwarded** and **X-Forwarded-For** HTTP headers per route. The values are: **append**: appends the header, preserving any existing header. This is the default value. **replace**: sets the header, removing any existing header. **never**: never sets the header, but preserves any existing header. **if-none**: sets the header if it is not already set.

1. By default, the router reloads every 5 s which resets the balancing connection across pods from the beginning. As a result, the **roundrobin** state is not preserved across reloads. This algorithm works best when pods have nearly identical computing capabilites and storage capacity. If your application or service has continuously changing endpoints, for example, due to the use of a CI/CD pipeline, uneven balancing can result. In this case, use a different algorithm.

2. If the number of IP addresses and CIDR ranges in an allowlist exceeds 61, they are written into a separate file that is then referenced from the **haproxy.config** file. This file is stored in the **/var/lib/haproxy/router/allowlists** folder.

NOTE

To ensure that the addresses are written to the allowlist, check that the full list of CIDR ranges are listed in the Ingress Controller configuration file. The etcd object size limit restricts how large a route annotation can be. Because of this, it creates a threshold for the maximum number of IP addresses and CIDR ranges that you can include in an allowlist.

A route that allows only one specific IP address

```
metadata:
  annotations:
    haproxy.router.openshift.io/ip_allowlist: 192.168.1.10
```

A route that allows several IP addresses

```
metadata:
  annotations:
    haproxy.router.openshift.io/ip_allowlist: 192.168.1.10 192.168.1.11 192.168.1.12
```

A route that allows an IP address CIDR network

```
metadata:
  annotations:
    haproxy.router.openshift.io/ip_allowlist: 192.168.1.0/24
```

A route that allows both IP an address and IP address CIDR networks

```
metadata:
  annotations:
    haproxy.router.openshift.io/ip_allowlist: 180.5.61.153 192.168.1.0/24 10.0.0.0/8
```

A route specifying a rewrite target

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  annotations:
    haproxy.router.openshift.io/rewrite-target: / (1)
...
```

(1) Sets / as rewrite path of the request on the backend.

Setting the **haproxy.router.openshift.io/rewrite-target** annotation on a route specifies that the Ingress Controller should rewrite paths in HTTP requests using this route before forwarding the requests to the backend application. The part of the request path that matches the path specified in **spec.path** is replaced with the rewrite target specified in the annotation.

The following table provides examples of the path rewriting behavior for various combinations of **spec.path**, request path, and rewrite target.

Table 1.4. rewrite-target examples

Route.spec.path	Request path	Rewrite target	Forwarded request path
/foo	/foo	/	/

Route.spec.path	Request path	Rewrite target	Forwarded request path
/foo	/foo/	/	/
/foo	/foo/bar	/	/bar
/foo	/foo/bar/	/	/bar/
/foo	/foo	/bar	/bar
/foo	/foo/	/bar	/bar/
/foo	/foo/bar	/baz	/baz/bar
/foo	/foo/bar/	/baz	/baz/bar/
/foo/	/foo	/	N/A (request path does not match route path)
/foo/	/foo/	/	/
/foo/	/foo/bar	/	/bar

Certain special characters in **haproxy.router.openshift.io/rewrite-target** require special handling because they must be escaped properly. Refer to the following table to understand how these characters are handled.

Table 1.5. Special character handling

For character	Use characters	Notes
#	\#	Avoid # because it terminates the rewrite expression
%	% or %%	Avoid odd sequences such as %%%
‘	\’	Avoid ‘ because it is ignored

All other valid URL characters can be used without escaping.

1.3.6. Throughput issue troubleshooting methods

To diagnose and resolve network throughput issues, such as unusually high latency between specific services, apply troubleshooting methods. Identifying connectivity bottlenecks helps ensure stable application performance within OpenShift Container Platform.

If pod logs do not reveal any cause of the problem, use the following methods to analyze performance issues:

- Use a packet analyzer, such as **ping** or **tcpdump** to analyze traffic between a pod and its node. For example, run the **tcpdump** tool on each pod while reproducing the behavior that led to the issue. Review the captures on both sides to compare send and receive timestamps to analyze the latency of traffic to and from a pod. Latency can occur in OpenShift Container Platform if a node interface is overloaded with traffic from other pods, storage devices, or the data plane.

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

  where:

  podip

  Specifies the IP address for the pod. Run the **oc get pod <pod_name> -o wide** command to get the IP address of a pod.
  The **tcpdump** command generates a file at **/tmp/dump.pcap** containing all traffic between these two pods. You can run the analyzer shortly before the issue is reproduced and stop the analyzer shortly after the issue is finished reproducing to minimize the size of the file. You can also run a packet analyzer between the nodes with:

  ```
  $ tcpdump -s 0 -i any -w /tmp/dump.pcap port 4789
  ```

- Use a bandwidth measuring tool, such as **iperf**, to measure streaming throughput and UDP throughput. Locate any bottlenecks by running the tool from the pods first, and then running it from the nodes.
  - For information on installing and using **iperf**, see this Red Hat Solution.
- In some cases, the cluster might mark the node with the router pod as unhealthy due to latency issues. Use worker latency profiles to adjust the frequency that the cluster waits for a status update from the node before taking action.
- If your cluster has designated lower-latency and higher-latency nodes, configure the **spec.nodePlacement** field in the Ingress Controller to control the placement of the router pod.

1.3.7. Configuring the route admission policy

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

1.3.8. Configuring the OpenShift Container Platform Ingress Controller for dual-stack networking

If your OpenShift Container Platform cluster is configured for IPv4 and IPv6 dual-stack networking, your cluster is externally reachable by OpenShift Container Platform routes.

The Ingress Controller automatically serves services that have both IPv4 and IPv6 endpoints, but you can configure the Ingress Controller for single-stack or dual-stack services.

Prerequisites

- You deployed an OpenShift Container Platform cluster on bare metal.
- You installed the OpenShift CLI (**oc**).

Procedure

1. To have the Ingress Controller serve traffic over IPv4/IPv6 to a workload, you can create a service YAML file or modify an existing service YAML file by setting the **ipFamilies** and **ipFamilyPolicy** fields. For example:

Sample service YAML file

Sample service YAML file

```
apiVersion: v1
kind: Service
metadata:
  creationTimestamp: yyyy-mm-ddT00:00:00Z
  labels:
    name: <service_name>
    manager: kubectl-create
    operation: Update
    time: yyyy-mm-ddT00:00:00Z
  name: <service_name>
  namespace: <namespace_name>
  resourceVersion: "<resource_version_number>"
  selfLink: "/api/v1/namespaces/<namespace_name>/services/<service_name>"
  uid: <uid_number>
spec:
  clusterIP: 172.30.0.0/16
  clusterIPs: 1
  - 172.30.0.0/16
  - <second_IP_address>
  ipFamilies: 2
  - IPv4
  - IPv6
  ipFamilyPolicy: RequireDualStack 3
  ports:
  - port: 8080
    protocol: TCP
    targetport: 8080
  selector:
    name: <namespace_name>
  sessionAffinity: None
  type: ClusterIP
status:
  loadbalancer: {}
```

1 1 In a dual-stack instance, there are two different **clusterIPs** provided.

2 For a single-stack instance, enter **IPv4** or **IPv6**. For a dual-stack instance, enter both **IPv4** and **IPv6**.

3 For a single-stack instance, enter **SingleStack**. For a dual-stack instance, enter **RequireDualStack**.

These resources generate corresponding **endpoints**. The Ingress Controller now watches **endpointslices**.

2. To view **endpoints**, enter the following command:

```
$ oc get endpoints
```

3. To view **endpointslices**, enter the following command:

```
$ oc get endpointslices
```

1.4. CREATING ADVANCED ROUTES

To secure application traffic and serve custom certificates to clients, configure routes by using edge, passthrough, or re-encrypt TLS termination. By using these methods, you can define granular encryption rules, ensuring that traffic is decrypted and re-encrypted according to your specific security requirements.

1.4.1. Creating an edge route with a custom certificate

To secure traffic by using a custom certificate, configure a route with edge TLS termination by running the **oc create route** command. This configuration terminates encryption at the Ingress Controller before forwarding traffic to the destination pod.

The route specifies the TLS certificate and key that the Ingress Controller uses for the route.

Prerequisites

- You must have a certificate/key pair in PEM-encoded files, where the certificate is valid for the route host.
- You may have a separate CA certificate in a PEM-encoded file that completes the certificate chain.

NOTE

Password protected key files are not supported. To remove a passphrase from a key file, use the following command:

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

Procedure

This procedure creates a **Route** resource with a custom certificate and edge TLS termination. The following assumes that the certificate/key pair are in the **tls.crt** and **tls.key** files in the current working directory. You may also specify a CA certificate if needed to complete the certificate chain. Substitute the actual path names for **tls.crt**, **tls.key**, and (optionally) **ca.crt**. Substitute the name of the service that you want to expose for **frontend**. Substitute the appropriate hostname for **www.example.com**.

- Create a secure **Route** resource using edge TLS termination and a custom certificate.

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

If you examine the resulting **Route** resource, the resource should have a configuration similar to the following example:

YAML Definition of the Secure Route

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: frontend
spec:
  host: www.example.com
```

```
  to:
    kind: Service
    name: frontend
  tls:
    termination: edge
    key: |-
      -----BEGIN PRIVATE KEY-----
      [...]
      -----END PRIVATE KEY-----
    certificate: |-
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
    caCertificate: |-
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
# ...
```

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

1.4.2. Creating a re-encrypt route with a custom certificate

To secure traffic by using a custom certificate, configure a route with re-encrypt TLS termination by running the **oc create route** command. This configuration enables the Ingress Controller to decrypt traffic, and then re-encrypt traffic before forwarding the traffic to the destination pod.

Prerequisites

- You must have a certificate/key pair in PEM-encoded files, where the certificate is valid for the route host.
- You may have a separate CA certificate in a PEM-encoded file that completes the certificate chain.
- You must have a separate destination CA certificate in a PEM-encoded file.
- You must have a service that you want to expose.

NOTE

Password protected key files are not supported. To remove a passphrase from a key file, use the following command:

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

Procedure

This procedure creates a **Route** resource with a custom certificate and reencrypt TLS termination. The following assumes that the certificate/key pair are in the **tls.crt** and **tls.key** files in the current working directory. You must also specify a destination CA certificate to enable the Ingress Controller to trust the service's certificate. You may also specify a CA certificate if needed to complete the certificate chain.

Substitute the actual path names for **tls.crt**, **tls.key**, **cacert.crt**, and (optionally) **ca.crt**. Substitute the name of the **Service** resource that you want to expose for **frontend**. Substitute the appropriate hostname for **www.example.com**.

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

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

If you examine the resulting **Route** resource, the resource should have a configuration similar to the following example:

YAML Definition of the Secure Route

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

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

1.4.3. Creating a passthrough route

To send encrypted traffic directly to the destination without decryption at the router, configure a route with passthrough termination by running the **oc create route** command. This configuration requires no key or certificate on the route, as the destination pod handles TLS termination.

Prerequisites

- You must have a service that you want to expose.

Procedure

- Create a **Route** resource:

```
$ oc create route passthrough route-passthrough-secured --service=frontend --port=8080
```

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

A Secured Route Using Passthrough Termination

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: route-passthrough-secured
spec:
  host: www.example.com
  port:
    targetPort: 8080
  tls:
    termination: passthrough
    insecureEdgeTerminationPolicy: None
  to:
    kind: Service
    name: frontend
```

where:

metadata.name

Specifies the name of the object, which is limited to 63 characters.

tls.termination

Specifies the **termination** field is set to **passthrough**. This is the only required **tls** field.

tls.insecureEdgeTerminationPolicy

Specifies the type of edge termination policy. Optional parameter. The only valid values are **None**, **Redirect**, or empty for disabled.
The destination pod is responsible for serving certificates for the traffic at the endpoint. This is currently the only method that can support requiring client certificates, also known as two-way authentication.

1.4.4. Creating a route using the destination CA certificate in the Ingress annotation

To define a route with a custom destination CA certificate, apply the **route.openshift.io/destination-ca-certificate-secret** annotation to an Ingress object. This configuration ensures the Ingress Controller uses the specified secret to verify the identity of the destination service.

Prerequisites

- You have a certificate/key pair in PEM-encoded files, where the certificate is valid for the route host.
- You have a separate CA certificate in a PEM-encoded file that completes the certificate chain.
- You have a separate destination CA certificate in a PEM-encoded file.

- You have a service that you want to expose.

Procedure

1. Create a secret for the destination CA certificate by entering the following command:

```
$ oc create secret generic dest-ca-cert --from-file=tls.crt=<file_path>
```

For example:

```
$ oc -n test-ns create secret generic dest-ca-cert --from-file=tls.crt=tls.crt
```

Example output

```
secret/dest-ca-cert created
```

2. Add the **route.openshift.io/destination-ca-certificate-secret** to the Ingress annotations:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: frontend
  annotations:
    route.openshift.io/termination: "reencrypt"
    route.openshift.io/destination-ca-certificate-secret: secret-ca-cert
...
```

where:

destination-ca-certificate-secret

Specifies the **route.openshift.io/destination-ca-certificate-secret** annotation. The annotation references a Kubernetes secret.
The Ingress Controller inserts a secret that is referenced in the annotation into the generated route.

Example output

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: frontend
  annotations:
    route.openshift.io/termination: reencrypt
    route.openshift.io/destination-ca-certificate-secret: secret-ca-cert
spec:
...
  tls:
    insecureEdgeTerminationPolicy: Redirect
    termination: reencrypt
    destinationCACertificate: |
      -----BEGIN CERTIFICATE-----
```

```
    [...]
    -----END CERTIFICATE-----
...
```

1.4.5. Creating a route with externally managed certificate

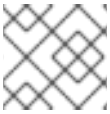
IMPORTANT

Securing route with external certificates in TLS secrets 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 .

You can configure OpenShift Container Platform routes with third-party certificate management solutions by using the **.spec.tls.externalCertificate** field of the route API. You can reference externally managed TLS certificates via secrets, eliminating the need for manual certificate management.

By using the externally managed certificate, you can reduce errors to ensure a smoother rollout of certificate updates and enable the OpenShift router to serve renewed certificates promptly. You can use externally managed certificates with both edge routes and re-encrypt routes.



NOTE

This feature applies to both edge routes and re-encrypt routes.

Prerequisites

- You must enable the **RouteExternalCertificate** feature gate.
- You have **create** permission on the **routes/custom-host** sub-resource, which is used for both creating and updating routes.
- You must have a secret containing a valid certificate or key pair in PEM-encoded format of type **kubernetes.io/tls**, which includes both **tls.key** and **tls.crt** keys. Example command: **$ oc create secret tls myapp-tls --cert=server.crt --key=server.key**.

Procedure

1. Create a **role** object in the same namespace as the secret to allow the router service account read access by running the following command:

```
$ oc create role secret-reader --verb=get,list,watch --resource=secrets --resource-name=
<secret-name> \
--namespace=<current-namespace>
```

   - **<secret-name>**: Specify the actual name of your secret.

- **<current-namespace>**: Specify the namespace where both your secret and route reside.

2. Create a **rolebinding** object in the same namespace as the secret and bind the router service account to the newly created role by running the following command:

```
$ oc create rolebinding secret-reader-binding --role=secret-reader --
serviceaccount=openshift-ingress:router --namespace=<current-namespace>
```

- **<current-namespace>**: Specify the namespace where both your secret and route reside.

3. Create a YAML file that defines the **route** and specifies the secret containing your certificate using the following example.

YAML definition of the secure route

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: myedge
  namespace: test
spec:
  host: myedge-test.apps.example.com
  tls:
    externalCertificate:
      name: <secret-name>
    termination: edge
    [...]
[...]
```

- **<secret-name>**: Specify the actual name of your secret.

4. Create a **route** resource by running the following command. If the secret exists and has a certificate/key pair, the router serves the generated certificate if all prerequisites are met.

```
$ oc apply -f <route.yaml> 1
```

- **<route.yaml>**: Specify the generated YAML filename.

NOTE

If **.spec.tls.externalCertificate** is not provided, the router uses default generated certificates.

You cannot provide the **.spec.tls.certificate** field or the **.spec.tls.key** field when using the **.spec.tls.externalCertificate** field.

1.4.6. Creating a route using the default certificate through an Ingress object

To generate a secure, edge-terminated route that uses the default ingress certificate, specify an empty TLS configuration in the Ingress object. This configuration overrides the default behavior, preventing the creation of an insecure route.

Prerequisites

- You have a service that you want to expose.
- You have access to the OpenShift CLI (**oc**).

Procedure

1. Create a YAML file for the Ingress object. In the following example, the file is called **example-ingress.yaml**:

   #### YAML definition of an Ingress object

   ```
   apiVersion: networking.k8s.io/v1
   kind: Ingress
   metadata:
     name: frontend
     ...
   spec:
     rules:
       ...
     tls:
     - {}
   ```

   where:

   spec.tls

   Specifies the TLS configuration. Use the exact syntax shown to specify TLS without specifying a custom certificate.

2. Create the Ingress object by running the following command:

   ```
   $ oc create -f example-ingress.yaml
   ```

Verification

- Verify that OpenShift Container Platform has created the expected route for the Ingress object by running the following command:

  ```
  $ oc get routes -o yaml
  ```

  #### Example output

  ```
  apiVersion: v1
  items:
  - apiVersion: route.openshift.io/v1
    kind: Route
    metadata:
      name: frontend-j9sdd
  # ...
    spec:
    ...
      tls:
        insecureEdgeTerminationPolicy: Redirect
        termination: edge
  # ...
  ```

where:

metadata.name

Specifies the name of the route, which includes the name of the Ingress object followed by a random suffix.

spec.tls

To use the default certificate, the route should not specify **spec.certificate**.

tls.termination

Specifies the termination policy for the route. The route should specify the **edge** termination policy.

CHAPTER 2. CONFIGURING INGRESS CLUSTER TRAFFIC

2.1. CONFIGURING INGRESS CLUSTER TRAFFIC OVERVIEW

To enable communication between external networks and services in OpenShift Container Platform, configure ingress cluster traffic.

2.1.1. Methods for communicating from outside the cluster

To enable communication between external networks and services in OpenShift Container Platform, configure the appropriate ingress method.

OpenShift Container Platform provides the following methods for communicating from outside the cluster with services running in the cluster. Note that the methods are listed in order of preference.

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

Method	Purpose
Use an Ingress Controller	Allows access to HTTP/HTTPS traffic and TLS-encrypted protocols other than HTTPS (for example, TLS with the SNI header).
Automatically assign an external IP using a load balancer service	Allows traffic to non-standard ports through an IP address assigned from a pool. Most cloud platforms offer a method to start a service with a load-balancer IP address.
About MetalLB and the MetalLB Operator	Allows traffic to a specific IP address or address from a pool on the machine network. For bare-metal installations or platforms that are like bare metal, MetalLB provides a way to start a service with a load-balancer IP address.
Manually assign an external IP to a service	Allows traffic to non-standard ports through a specific IP address.
Configure a **NodePort**	Expose a service on all nodes in the cluster.

2.1.2. Additional resources

- Use an Ingress Controller
- Automatically assign an external IP using a load balancer service
- About MetalLB and the MetalLB Operator

- Manually assign an external IP to a service
- Configure a **NodePort**

2.1.3. Comparison: Fault-tolerant access to external IP addresses

To ensure continuous service availability and maintain external IP access in OpenShift Container Platform, configure fault-tolerant networking features.

For the communication methods that provide access to an external IP address, fault tolerant access to the IP address is another consideration. The following features provide fault tolerant access to an external IP address.

IP failover

IP failover manages a pool of virtual IP addresses for a set of nodes. IP failover gets implemented with Keepalived and Virtual Router Redundancy Protocol (VRRP). IP failover is a layer 2 mechanism only and relies on multicast. Multicast can have disadvantages for some networks.

MetalLB

MetalLB has a layer 2 mode, but it does not use multicast. Layer 2 mode has a disadvantage that it transfers all traffic for an external IP address through one node.

Manually assigning external IP addresses

You can configure your cluster with an IP address block that is used to assign external IP addresses to services. By default, this feature is disabled. This feature is flexible, but places the largest burden on the cluster or network administrator. The cluster is prepared to receive traffic that is destined for the external IP, but you must decide how to route traffic to nodes.

2.2. CONFIGURING EXTERNALIPS FOR SERVICES

As a cluster administrator, you can select an IP address block that is external to the cluster and can send traffic to services in the cluster. This functionality is generally most useful for clusters installed on bare-metal hardware.

IMPORTANT

Before you configure ExternalIPs for services, your network infrastructure must route traffic for the external IP addresses to your cluster.

2.2.1. About ExternalIP

To load balance traffic in non-cloud environments, use the ExternalIP facility to specify external IP addresses in the **spec.externalIPs[]** parameter of the **Service** object. This configuration directs traffic to a local node, providing functionality similar to a **type=NodePort** service.

IMPORTANT

For cloud environments, use the load balancer services for automatic deployment of a cloud load balancer to target the endpoints of a service.

After you specify a value for the parameter, OpenShift Container Platform assigns an additional virtual IP address to the service. The IP address can exist outside of the service network that you defined for your cluster.

WARNING

Because ExternalIP is disabled by default, enabling the ExternalIP functionality might introduce security risks for the service, because in-cluster traffic to an external IP address is directed to that service. This configuration means that cluster users could intercept sensitive traffic destined for external resources.

You can use either a MetalLB implementation or an IP failover deployment to attach an ExternalIP resource to a service in the following ways:

Automatic assignment of an external IP

OpenShift Container Platform automatically assigns an IP address from the **autoAssignCIDRs** CIDR block to the **spec.externalIPs[]** array when you create a **Service** object with **spec.type=LoadBalancer** set. For this configuration, OpenShift Container Platform implements a cloud version of the load balancer service type and assigns IP addresses to the services. Automatic assignment is disabled by default and must be configured by a cluster administrator as described in the "Configuration for ExternalIP" section.

Manual assignment of an external IP

OpenShift Container Platform uses the IP addresses assigned to the **spec.externalIPs[]** array when you create a **Service** object. You cannot specify an IP address that is already in use by another service.

After using either the MetalLB implementation or an IP failover deployment to host external IP address blocks, you must configure your networking infrastructure to ensure that the external IP address blocks are routed to your cluster. This configuration means that the IP address is not configured in the network interfaces from nodes. To handle the traffic, you must configure the routing and access to the external IP by using a method, such as static Address Resolution Protocol (ARP) entries.

OpenShift Container Platform extends the ExternalIP functionality in Kubernetes by adding the following capabilities:

- Restrictions on the use of external IP addresses by users through a configurable policy
- Allocation of an external IP address automatically to a service upon request

2.2.2. Configuration for ExternalIP

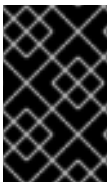
You can set parameters in the **Network.config.openshift.io** custom resource (CR) to govern the use of an external IP address in OpenShift Container Platform.

The following list details these parameters:

- **spec.externalIP.autoAssignCIDRs** defines an IP address block used by the load balancer when choosing an external IP address for the service. OpenShift Container Platform supports only a single IP address block for automatic assignment. This configuration requires less steps than manually assigning ExternalIPs to services, which requires managing the port space of a limited number of shared IP addresses. If you enable automatic assignment, the Cloud Controller Manager Operator allocates an external IP address to a **Service** object with **spec.type=LoadBalancer** defind in its configuration.

- **spec.externalIP.policy** defines the permissible IP address blocks when manually specifying an IP address. OpenShift Container Platform does not apply policy rules to IP address blocks that you defined in the **spec.externalIP.autoAssignCIDRs** parameter.

If routed correctly, external traffic from the configured external IP address block can reach service endpoints through any TCP or UDP port that the service exposes.



IMPORTANT

As a cluster administrator, you must configure routing to externalIPs. You must also ensure that the IP address block you assign terminates at one or more nodes in your cluster. For more information, see Kubernetes External IPs .

OpenShift Container Platform supports both automatic and manual IP address assignment. This support guarantees that each address gets assigned to a maximum of one service and that each service can expose its chosen ports regardless of the ports exposed by other services.

NOTE

To use IP address blocks defined by **autoAssignCIDRs** in OpenShift Container Platform, you must configure the necessary IP address assignment and routing for your host network.

The following YAML describes a service with an external IP address configured:

Example Service object with spec.externalIPs[] set

```
apiVersion: v1
kind: Service
metadata:
  name: http-service
spec:
  clusterIP: 172.30.163.110
  externalIPs:
  - 192.168.132.253
  externalTrafficPolicy: Cluster
  ports:
  - name: highport
    nodePort: 31903
    port: 30102
    protocol: TCP
    targetPort: 30102
  selector:
    app: web
  sessionAffinity: None
  type: LoadBalancer
status:
  loadBalancer:
    ingress:
    - ip: 192.168.132.253
# ...
```

If you run a private cluster on a cloud-provider platform, you can change the publishing scope to **internal** for the load balancer of the Ingress Controller by running the following **patch** command:

```
$ oc -n openshift-ingress-operator patch ingresscontrollers/ingress-controller-with-nlb --type=merge --patch='{"spec":{"endpointPublishingStrategy":{"loadBalancer":{"scope":"Internal"}}}}'
```

After you run this command, the Ingress Controller restricts access to routes for OpenShift Container Platform applications to internal networks only.

2.2.3. Applying restrictions on the assignment of an external IP address

As a cluster administrator, you can specify IP address blocks to allow and to reject IP addresses for a service. Restrictions apply only to users without **cluster-admin** privileges. A cluster administrator can always set the service **spec.externalIPs[]** field to any IP address.

When configuring policy restrictions, the following rules apply:

- If **policy** is set to **{}**, creating a **Service** object with **spec.ExternalIPs[]** results in a failed service. This setting is the default for OpenShift Container Platform. The same behavior exists for **policy: null**.
- If **policy** is set and either **policy.allowedCIDRs[]** or **policy.rejectedCIDRs[]** is set, the following rules apply:
  - If **allowedCIDRs[]** and **rejectedCIDRs[]** are both set, **rejectedCIDRs[]** has precedence over **allowedCIDRs[]**.
  - If **allowedCIDRs[]** is set, creating a **Service** object with **spec.ExternalIPs[]** succeeds only if the specified IP addresses are allowed.
  - If **rejectedCIDRs[]** is set, creating a **Service** object with **spec.ExternalIPs[]** succeeds only if the specified IP addresses are not rejected.

Procedure

- Configure an IP address policy by specifying Classless Inter-Domain Routing (CIDR) address blocks for the **spec.ExternalIP.policy** parameter in the **policy** object:

  Example in JSON form of a policy object and its CIDR parameters

```
{
  "policy": {
    "allowedCIDRs": [],
    "rejectedCIDRs": []
  }
}
```

2.2.4. Example policy objects

Reference the examples in the **Example policy objects** section to understand different **spec.externalIP.policy** configurations.

In the following example, the policy prevents OpenShift Container Platform from creating any service with a specified external IP address.

Example policy to reject any value specified for Service object spec.externalIPs[]

-

```
apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  externalIP:
    policy: {}
# ...
```

In the following example, both the **allowedCIDRs** and **rejectedCIDRs** fields are set.

Example policy that includes both allowed and rejected CIDR blocks

```
apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  externalIP:
    policy:
      allowedCIDRs:
      - 172.16.66.10/23
      rejectedCIDRs:
      - 172.16.66.10/24
# ...
```

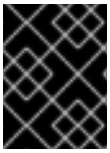
In the following example, **policy** is set to **{}**. With this configuration, using the **oc get networks.config.openshift.io -o yaml** command to view the configuration means **policy** parameter does not show on the command output. The same behavior exists for **policy: null**.

Example policy to allow any value specified for Service object spec.externalIPs[]

```
apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  clusterNetwork:
  - cidr: 10.128.0.0/14
    hostPrefix: 23
  externalIP:
    policy: {}
# ...
```

2.2.5. ExternalIP address block configuration

To better understand ExternalIP address blocks, view the example configuration for ExternalIP address blocks that is defined by a Network custom resource (CR) named **cluster**. The Network CR is part of the **config.openshift.io** API group.



IMPORTANT

During cluster installation, the Cluster Version Operator (CVO) automatically creates a Network CR named **cluster**. Creating any other CR objects of this type is not supported.

The following YAML describes the ExternalIP configuration in a **Network.config.openshift.io** CR named **cluster**:

```
apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  externalIP:
    autoAssignCIDRs: []
    policy:
      ...
```

- **autoAssignCIDRs**: Defines the IP address block in CIDR format that is available for automatic assignment of external IP addresses to a service. Only a single IP address range is allowed.
- **policy**: Defines restrictions on manual assignment of an IP address to a service. If no restrictions are defined, specifying the **spec.externalIP** field in a **Service** object is not allowed. By default, no restrictions are defined.

The following YAML describes the fields for the **policy** stanza in the **Network.config.openshift.io** CR:

```
policy:
  allowedCIDRs: []
  rejectedCIDRs: []
```

- **allowedCIDRs**: A list of allowed IP address ranges in CIDR format.
- **rejectedCIDRs**: A list of rejected IP address ranges in CIDR format.

The next set of example configurations show external IP address pools configurations.

The following YAML shows a **spec.externalIP.autoAssignCIDRs** configuration that enables automatically assigned external IP addresses:

Example configuration with spec.externalIP.autoAssignCIDRs set

```
apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  ...
  externalIP:
    autoAssignCIDRs:
    - 192.168.132.254/29
```

The following YAML configuration includes a **spec.externalIP.policy** configuration that sets policy rules for the allowed and rejected CIDR ranges:

```
apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  ...
  externalIP:
    policy:
      allowedCIDRs:
      - 192.168.132.0/29
      - 192.168.132.8/29
      rejectedCIDRs:
      - 192.168.132.7/32
```

2.2.6. Configure external IP address blocks for your cluster

As a cluster administrator, you can configure the ExternalIP settings to provide predictable entry points for external traffic to reach your cluster.

The following list details these ExternalIP settings:

- An ExternalIP address block used by OpenShift Container Platform to automatically populate the **spec.clusterIP** field for a **Service** object.
- A policy object to restrict what IP addresses may be manually assigned to the **spec.clusterIP** array of a **Service** object.

Prerequisites

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

Procedure

1. Optional: To display the current external IP configuration, enter the following command:

```
$ oc describe networks.config cluster
```

2. To edit the configuration, enter the following command:

```
$ oc edit networks.config cluster
```

3. Modify the ExternalIP configuration, as in the following example:

```
apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  ...
  externalIP:
  ...
```

- **externalIP**: Specify the configuration for the **externalIP** stanza.

4. To confirm the updated ExternalIP configuration, enter the following command:

```
$ oc get networks.config cluster -o go-template='{{.spec.externalIP}}{{"\n"}}'
```

2.2.7. Additional resources

- Configuring IP failover
- About MetalLB and the MetalLB Operator

2.2.8. Next steps

- Configuring ingress cluster traffic for a service external IP

2.3. CONFIGURING INGRESS CLUSTER TRAFFIC BY USING AN INGRESS CONTROLLER

You can use the Ingress Controller to control how external users communicate with services that run inside the cluster.

Before you begin any of the procedures that are listed in the Configuring ingress cluster traffic by using an Ingress Controller document, ensure that you meet the following prerequisites. A cluster administrator performs these prerequisites:

- Set up the external port to the cluster networking environment so that requests can reach the cluster.
- Make sure there is at least one user with cluster admin role. To add this role to a user, run the following command:

```
$ oc adm policy add-cluster-role-to-user cluster-admin username
```

- You have an OpenShift Container Platform cluster with at least one master and at least one node and a system outside the cluster that has network access to the cluster. This procedure assumes that the external system is on the same subnet as the cluster. The additional networking required for external systems on a different subnet is out-of-scope for this topic.

2.3.1. Using Ingress Controllers and routes

You can use the Ingress Controller to allow external access to an OpenShift Container Platform cluster. The Ingress Operator manages Ingress Controllers and wildcard DNS.

An Ingress Controller is configured to accept external requests and proxy them based on the configured routes. This is limited to HTTP, HTTPS using SNI, and TLS using SNI, which is sufficient for web applications and services that work over TLS with SNI.

Work with your administrator to configure an Ingress Controller to accept external requests and proxy them based on the configured routes.

The administrator can create a wildcard DNS entry and then set up an Ingress Controller. Then, you can work with the edge Ingress Controller without having to contact the administrators.

By default, every Ingress Controller in the cluster can admit any route created in any project in the cluster. The Ingress Controller has the following characteristics:

- Has two replicas by default, which means it should be running on two compute nodes.
- Can be scaled up to have more replicas on more nodes.

2.3.2. Creating a project and service

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

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

Prerequisites

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

Procedure

1. Create a new project for your service by running the **oc new-project** command:

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

2. Use the **oc new-app** command to create your service:

```
$ oc new-app nodejs:12~https://github.com/sclorg/nodejs-ex.git
```

3. To verify that the service was created, run the following command:

```
$ oc get svc -n <project_name>
```

Example output

```
NAME       TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
nodejs-ex  ClusterIP  172.30.197.157  <none>        8080/TCP   70s
```

NOTE

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

2.3.3. Exposing the service by creating a route

To enable external access to your application that runs on OpenShift Container Platform, you can expose the service as a route by using the **oc expose** command.

Prerequisites

- You logged into OpenShift Container Platform.

Procedure

1. Log in to the project where the service you want to expose is located:

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

2. Run the **oc expose service** command to expose the route:

```
$ oc expose service nodejs-ex
```

Example output

```
route.route.openshift.io/nodejs-ex exposed
```

3. To verify that the service is exposed, you can use a tool, such as **curl** to check that the service is accessible from outside the cluster.

   a. To find the hostname of the route, enter the following command:

```
$ oc get route
```

   Example output

```
NAME       HOST/PORT                         PATH   SERVICES   PORT       TERMINATION   WILDCARD
nodejs-ex  nodejs-ex-myproject.example.com          nodejs-ex  8080-tcp                 None
```

   b. To check that the host responds to a GET request, enter the following command:

   Example curl command

```
$ curl --head nodejs-ex-myproject.example.com
```

   Example output

```
HTTP/1.1 200 OK
...
```

2.3.4. Ingress sharding in OpenShift Container Platform

To optimise routing performance in OpenShift Container Platform, create shards so that you can load balance incoming traffic across multiple Ingress Controllers.

In OpenShift Container Platform, an Ingress Controller can serve all routes, or it can serve a subset of routes. By default, the Ingress Controller serves any route created in any namespace of the cluster. You can add additional Ingress Controllers to your cluster to optimize routing by creating shards, which are subsets of routes based on selected characteristics. To mark a route as a member of a shard, use labels in the route or namespace **metadata** field. The Ingress Controller uses *selectors*, also known as a *selection expression*, to select a subset of routes from the entire pool of routes to serve.

You can also use Ingress sharding when you want to isolate traffic so that the traffic can route to a specific Ingress Controller.

By default, each route uses the default domain of the cluster. However, routes can be configured to use the domain of the router instead.

2.3.5. Ingress Controller sharding

You can use Ingress sharding, also known as *router sharding*, to distribute a set of routes across multiple routers by adding labels to routes, namespaces, or both.

The Ingress Controller uses a corresponding set of selectors to admit only the routes that have a specified label. Each Ingress shard comprises the routes that are filtered by using a given selection expression.

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

- Balance Ingress Controllers, or routers, with several routes to accelerate responses to changes.
- Assign certain routes to have different reliability guarantees than other routes.
- Allow certain Ingress Controllers to have different policies defined.
- Allow only specific routes to use additional features.
- Expose different routes on different addresses so that internal and external users can see different routes, for example.
- Transfer traffic from one version of an application to another during a blue-green deployment.

When Ingress Controllers are sharded, a given route is admitted to zero or more Ingress Controllers in the group. The status of a route describes whether an Ingress Controller has admitted the route. An Ingress Controller only admits a route if the route is unique to a shard.

With sharding, you can distribute subsets of routes over multiple Ingress Controllers. These subsets can be nonoverlapping, also called *traditional* sharding, or overlapping, otherwise known as *overlapped* sharding.

The following table outlines three sharding methods:

Sharding method	Description
Namespace selector	After you add a namespace selector to the Ingress Controller, all routes in a namespace that have matching labels for the namespace selector are included in the Ingress shard. Consider this method when an Ingress Controller serves all routes created in a namespace.
Route selector	After you add a route selector to the Ingress Controller, all routes with labels that match the route selector are included in the Ingress shard. Consider this method when you want an Ingress Controller to serve only a subset of routes or a specific route in a namespace.
Namespace and route selectors	Provides your Ingress Controller scope for both namespace selector and route selector methods. Consider this method when you want the flexibility of both the namespace selector and the route selector methods.

2.3.5.1. Traditional sharding example

To understand traditional sharding, you can review the example of a configured Ingress Controller **finops-router** that has the label selector **spec.namespaceSelector.matchExpressions** with key values set to **finance** and **ops**.

Example YAML definition for finops-router

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: finops-router
  namespace: openshift-ingress-operator
spec:
  namespaceSelector:
    matchExpressions:
    - key: name
      operator: In
      values:
      - finance
      - ops
```

An example of a configured Ingress Controller **dev-router** that has the label selector **spec.namespaceSelector.matchLabels.name** with the key value set to **dev**:

Example YAML definition for dev-router

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: dev-router
  namespace: openshift-ingress-operator
spec:
  namespaceSelector:
    matchLabels:
      name: dev
```

If all application routes are in separate namespaces, such as each labeled with **name:finance**, **name:ops**, and **name:dev**, the configuration effectively distributes your routes between the two Ingress Controllers. OpenShift Container Platform routes for console, authentication, and other purposes should not be handled.

In the previous scenario, sharding becomes a special case of partitioning, with no overlapping subsets. Routes are divided between router shards.

WARNING

The **default** Ingress Controller continues to serve all routes unless the **namespaceSelector** or **routeSelector** fields contain routes that are meant for exclusion. See this Red Hat Knowledgebase solution and the section "Sharding the default Ingress Controller" for more information on how to exclude routes from the default Ingress Controller.

2.3.5.2. Overlapped sharding example

An example of a configured Ingress Controller **devops-router** that has the label selector **spec.namespaceSelector.matchExpressions** with key values set to **dev** and **ops**:

Example YAML definition for devops-router

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: devops-router
  namespace: openshift-ingress-operator
spec:
  namespaceSelector:
    matchExpressions:
    - key: name
      operator: In
      values:
      - dev
      - ops
```

The routes in the namespaces labeled **name:dev** and **name:ops** are now serviced by two different Ingress Controllers. With this configuration, you have overlapping subsets of routes.

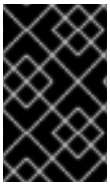
With overlapping subsets of routes you can create more complex routing rules. For example, you can divert higher priority traffic to the dedicated **finops-router** while sending lower priority traffic to **devops-router**.

2.3.5.3. Sharding the default Ingress Controller

You can restrict an Ingress Controller from servicing routes with specific labels by using either namespace selectors or route selectors.

After creating a new Ingress shard, there might be routes that are admitted to your new Ingress shard that are also admitted by the default Ingress Controller. This is because the default Ingress Controller has no selectors and admits all routes by default.

The following procedure restricts the default Ingress Controller from servicing your newly sharded **finance**, **ops**, and **dev**, routes by using a namespace selector. This adds further isolation to Ingress shards.



IMPORTANT

You must keep all of OpenShift Container Platform's administration routes on the same Ingress Controller. Therefore, avoid adding additional selectors to the default Ingress Controller that exclude these essential routes.

Prerequisites

- You installed the OpenShift CLI (**oc**).
- You are logged in as a project administrator.

Procedure

1. Modify the default Ingress Controller by running the following command:

```
$ oc edit ingresscontroller -n openshift-ingress-operator default
```

2. Edit the Ingress Controller to contain a **namespaceSelector** that excludes the routes with any of the **finance**, **ops**, and **dev** labels:

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  namespaceSelector:
    matchExpressions:
      - key: name
        operator: NotIn
        values:
          - finance
          - ops
          - dev
```

The default Ingress Controller no longer serves the namespaces labeled with **name:finance**, **name:ops**, and **name:dev**.

2.3.5.4. Ingress sharding and DNS

To ensure ingress traffic reaches the correct router in OpenShift Container Platform, create separate DNS entries for each router in your project. Because a router does not forward unknown traffic to other routers, distinct entries are required to resolve specific domains to their hosting nodes."

Consider the following example:

- Router A lives on host 192.168.0.5 and has routes with ***.foo.com**.
- Router B lives on host 192.168.1.9 and has routes with ***.example.com**.

Separate DNS entries must resolve ***.foo.com** to the node hosting Router A and ***.example.com** to the node hosting Router B:

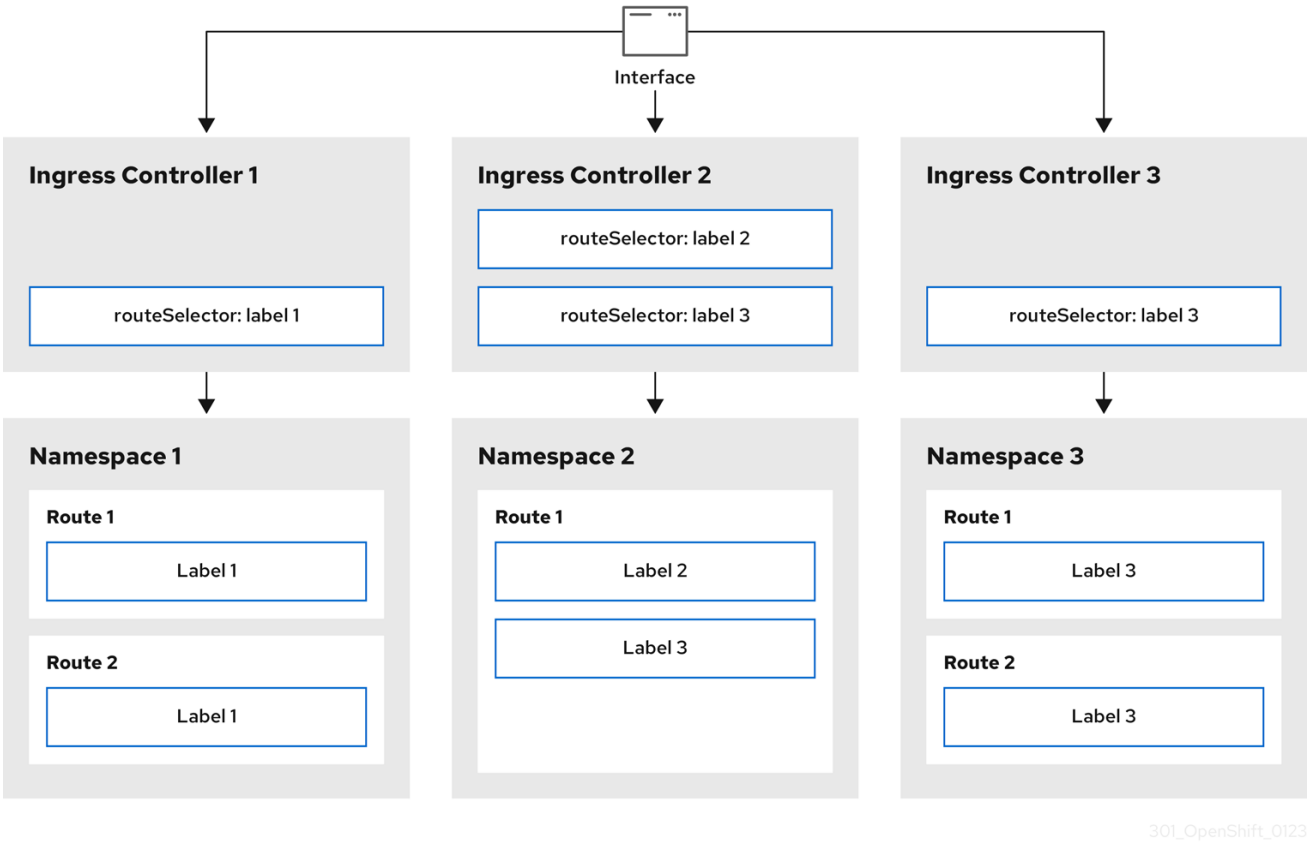
- ***.foo.com A IN 192.168.0.5**

- ***.example.com A IN 192.168.1.9**

2.3.5.5. Configuring Ingress Controller sharding by using route labels

You can use route labels to configure Ingress Controller sharding so that the Ingress Controller serves any route in any namespace that is selected by the route selector.

Figure 2.1. Ingress sharding by using route labels



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

Procedure

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

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: sharded
  namespace: openshift-ingress-operator
spec:
  domain: <apps-sharded.basedomain.example.net>
  nodePlacement:
    nodeSelector:
      matchLabels:
        node-role.kubernetes.io/worker: ""
```

```
  routeSelector:
    matchLabels:
      type: sharded
```

- **<apps-sharded.basedomain.example.net>**: Specify a domain to be used by the Ingress Controller. This domain must be different from the default Ingress Controller domain.

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

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

The Ingress Controller selects routes in any namespace that have the label **type: sharded**.

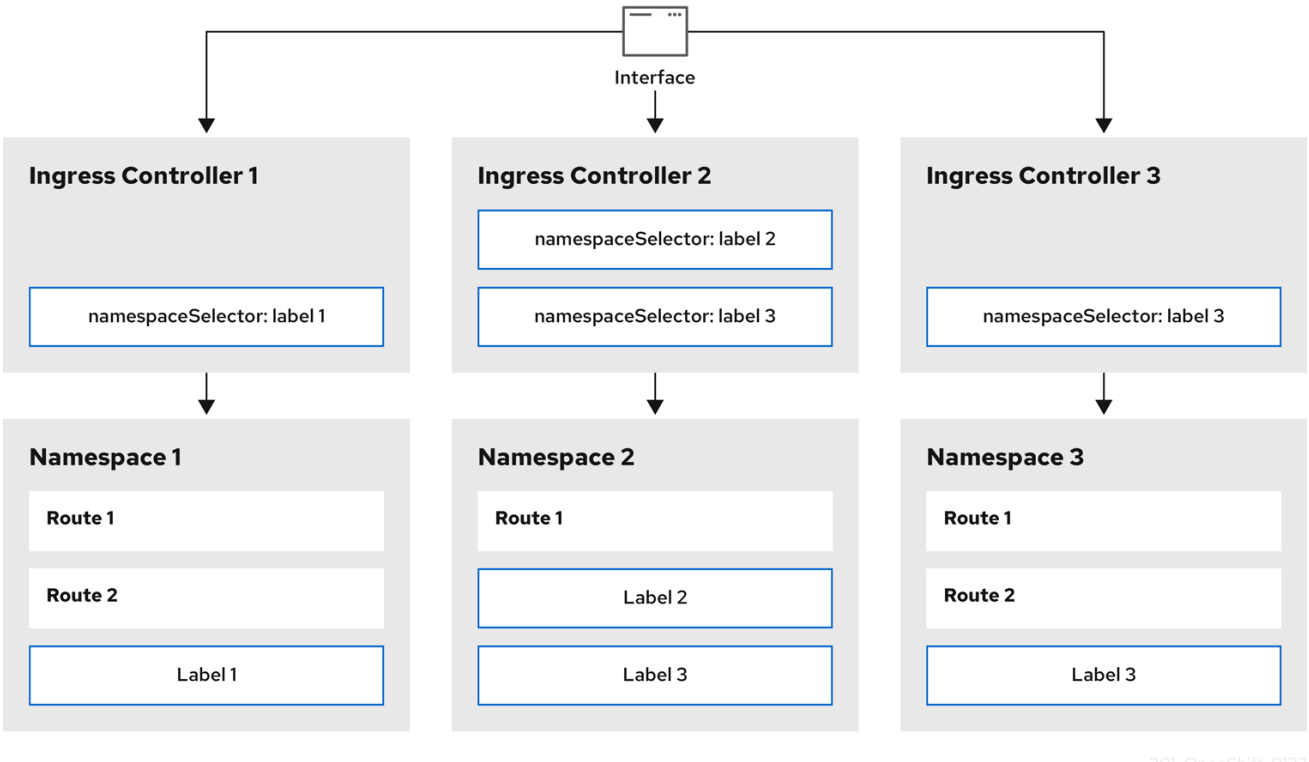
3. Create a new route by using the domain configured in the **router-internal.yaml**:

```
$ oc expose svc <service-name> --hostname <route-name>.apps-
sharded.basedomain.example.net
```

2.3.5.6. Configuring Ingress Controller sharding by using namespace labels

You can use namespace labels to configure Ingress Controller sharding so that the Ingress Controller serves any route in any namespace that is selected by the namespace selector.

Figure 2.2. Ingress sharding by using namespace labels



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

Procedure

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

```
$ cat router-internal.yaml
```

Example output

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: sharded
  namespace: openshift-ingress-operator
spec:
  domain: <apps-sharded.basedomain.example.net>
  nodePlacement:
    nodeSelector:
      matchLabels:
        node-role.kubernetes.io/worker: ""
  namespaceSelector:
    matchLabels:
      type: sharded
```

- **<apps-sharded.basedomain.example.net>**: Specify a domain to be used by the Ingress Controller. This domain must be different from the default Ingress Controller domain.

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

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

The Ingress Controller selects routes in any namespace that is selected by the namespace selector that have the label **type: sharded**.

3. Create a new route by using the domain configured in the **router-internal.yaml**:

```
$ oc expose svc <service-name> --hostname <route-name>.apps-
sharded.basedomain.example.net
```

2.3.5.7. Creating a route for Ingress Controller sharding

To host applications at specific URLs and balance traffic load in OpenShift Container Platform, configure Ingress Controller sharding. By sharding, you can isolate traffic for specific workloads or tenants, ensuring efficient resource management across your cluster.

The following procedure describes how to create a route for Ingress Controller sharding, using the **hello-openshift** application as an example.

Prerequisites

- You installed the OpenShift CLI (**oc**).
- You are logged in as a project administrator.
- You have a web application that exposes a port and an HTTP or TLS endpoint listening for traffic on the port.

- You have configured the Ingress Controller for sharding.

Procedure

1. Create a project called **hello-openshift** by running the following command:

```
$ oc new-project hello-openshift
```

2. Create a pod in the project by running the following command:

```
$ oc create -f https://raw.githubusercontent.com/openshift/origin/master/examples/hello-openshift/hello-pod.json
```

3. Create a service called **hello-openshift** by running the following command:

```
$ oc expose pod/hello-openshift
```

4. Create a route definition called **hello-openshift-route.yaml**:

YAML definition of the created route for sharding

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  labels:
    type: sharded
  name: hello-openshift-edge
  namespace: hello-openshift
spec:
  subdomain: hello-openshift
  tls:
    termination: edge
  to:
    kind: Service
    name: hello-openshift
```

where:

type

Specifies both the label key and its corresponding label value must match the ones specified in the Ingress Controller. In this example, the Ingress Controller has the label key and value **type: sharded**.

subdomain

Specifies the route gets exposed by using the value of the **subdomain** field. When you specify the **subdomain** field, you must leave the hostname unset. If you specify both the **host** and **subdomain** fields, then the route uses the value of the **host** field, and ignore the **subdomain** field.

5. Use **hello-openshift-route.yaml** to create a route to the **hello-openshift** application by running the following command:

```
$ oc -n hello-openshift create -f hello-openshift-route.yaml
```

Verification

- Get the status of the route with the following command:

```
$ oc -n hello-openshift get routes/hello-openshift-edge -o yaml
```

The resulting **Route** resource should look similar to the following:

Example output

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  labels:
    type: sharded
  name: hello-openshift-edge
  namespace: hello-openshift
spec:
  subdomain: hello-openshift
  tls:
    termination: edge
  to:
    kind: Service
    name: hello-openshift
status:
  ingress:
  - host: hello-openshift.<apps-sharded.basedomain.example.net>
    routerCanonicalHostname: router-sharded.<apps-sharded.basedomain.example.net>
    routerName: sharded
```

where:

host

Specifies the hostname the Ingress Controller, or router, uses to expose the route. The value of the **host** field is automatically determined by the Ingress Controller, and uses its domain. In this example, the domain of the Ingress Controller is **<apps-sharded.basedomain.example.net>**.

<apps-sharded.basedomain.example.net>

Specifies the hostname of the Ingress Controller. If the hostname is not set, the route can use a subdomain instead. When you specify a subdomain, you automatically use the domain of the Ingress Controller that exposes the route. When a route is exposed by multiple Ingress Controllers, the route is hosted at multiple URLs.

routerName

Specifies the name of the Ingress Controller. In this example, the Ingress Controller has the name **sharded**.

2.3.5.8. Additional resources

- Baseline Ingress Controller (router) performance
- Configuring the Ingress Controller
- Installing a cluster on bare metal

- Installing a cluster on vSphere
- About network policy

2.4. CONFIGURING THE INGRESS CONTROLLER ENDPOINT PUBLISHING STRATEGY

To expose Ingress Controller endpoints to external systems and enable load balancer integrations in OpenShift Container Platform, configure the **endpointPublishingStrategy** parameter.

IMPORTANT

On Red Hat OpenStack Platform (RHOSP), the **LoadBalancerService** endpoint publishing strategy is supported only if a cloud provider is configured to create health monitors. For RHOSP 16.2, this strategy is possible only if you use the Amphora Octavia provider.

For more information, see the "Setting RHOSP Cloud Controller Manager options" section of the RHOSP installation documentation.

2.4.1. Ingress Controller endpoint publishing strategy

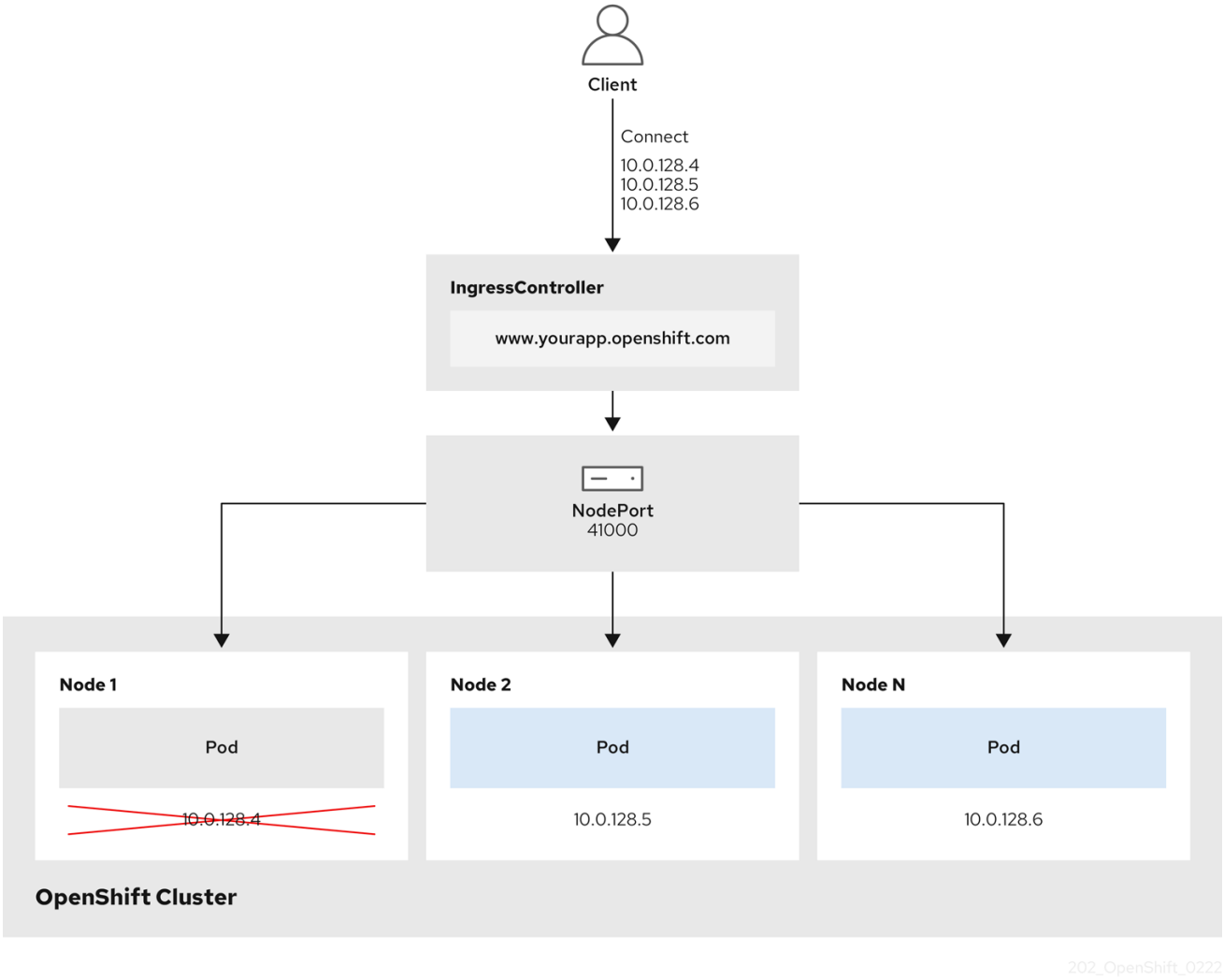
To expose Ingress Controller endpoints to external networks in OpenShift Container Platform, configure either the **NodePortService** endpoint publishing strategy or the **HostNetwork** endpoint publishing strategy.

NodePortService endpoint publishing strategy

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

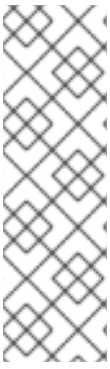
In this configuration, the Ingress Controller deployment uses container networking. A **NodePortService** is created to publish the deployment. The specific node ports are dynamically allocated by OpenShift Container Platform; however, to support static port allocations, your changes to the node port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

Figure 2.3. Diagram of NodePortService



The preceding graphic shows the following concepts pertaining to OpenShift Container Platform Ingress NodePort endpoint publishing strategy:

- All the available nodes in the cluster have their own, externally accessible IP addresses. The service running in the cluster is bound to the unique NodePort for all the nodes.
- When the client connects to a node that is down, for example, by connecting the **10.0.128.4** IP address in the graphic, the node port directly connects the client to an available node that is running the service. In this scenario, no load balancing is required. As the image shows, the **10.0.128.4** address is down and another IP address must be used instead.



NOTE

The Ingress Operator ignores any updates to **.spec.ports[].nodePort** fields of the service.

By default, ports are allocated automatically and you can access the port allocations for integrations. However, sometimes static port allocations are necessary to integrate with existing infrastructure which may not be easily reconfigured in response to dynamic ports. To achieve integrations with static node ports, you can update the managed service resource directly.

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

HostNetwork endpoint publishing strategy*

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

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

The **HostNetwork** object has a **hostNetwork** field with the following default values for the optional binding ports: **httpPort: 80**, **httpsPort: 443**, and **statsPort: 1936**. By specifying different binding ports for your network, you can deploy multiple Ingress Controllers on the same node for the **HostNetwork** strategy.

Example

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: internal
  namespace: openshift-ingress-operator
spec:
  domain: example.com
  endpointPublishingStrategy:
    type: HostNetwork
    hostNetwork:
      httpPort: 80
      httpsPort: 443
      statsPort: 1936
```

2.4.1.1. Configuring the Ingress Controller endpoint publishing scope to Internal

To restrict cluster access to internal traffic and enhance network security in OpenShift Container Platform, change the Ingress Controller scope from **External** to **Internal**.

When a cluster administrator installs a new cluster without specifying that the cluster is private, the default Ingress Controller is created with a **scope** set to **External**.

Prerequisites

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

Procedure

- To change an **External**-scoped Ingress Controller to **Internal**, enter the following command:

```
$ oc -n openshift-ingress-operator patch ingresscontrollers/default --type=merge --
patch='{"spec":{"endpointPublishingStrategy":{"type":"LoadBalancerService","loadBalancer":
{"scope":"Internal"}}}}'
```

Verification

- To check the status of the Ingress Controller, enter the following command:

```
$ oc -n openshift-ingress-operator get ingresscontrollers/default -o yaml
```

- The **Progressing** status condition indicates whether you must take further action. For example, the status condition can indicate that you need to delete the service by entering the following command:

```
$ oc -n openshift-ingress delete services/router-default
```

If you delete the service, the Ingress Operator recreates it as **Internal**.

2.4.1.2. Configuring the Ingress Controller endpoint publishing scope to External

To expose cluster services to public networks or the internet in OpenShift Container Platform, configure the Ingress Controller endpoint publishing scope to **External**.

When a cluster administrator installs a new cluster without specifying that the cluster is private, the default Ingress Controller is created with a **scope** set to **External**.

As an installation or post-installation task, a cluster administrator can configure the Ingress Controller to **Internal**. Additionally, a cluster administrator can change an **Internal** Ingress Controller to **External**.

IMPORTANT

On some platforms, it is necessary to delete and recreate the service.

Changing the scope can cause disruption to Ingress traffic, potentially for several minutes. This applies to platforms where it is necessary to delete and recreate the service, because the procedure can cause OpenShift Container Platform to deprovision the existing service load balancer, provision a new one, and update DNS.

Prerequisites

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

Procedure

- To change an **Internal**-scoped Ingress Controller to **External**, enter the following command:

```
$ oc -n openshift-ingress-operator patch ingresscontrollers/private --type=merge --
patch='{"spec":{"endpointPublishingStrategy":{"type":"LoadBalancerService","loadBalancer":
{"scope":"External"}}}}'
```

Verification

- To check the status of the Ingress Controller, enter the following command:

```
$ oc -n openshift-ingress-operator get ingresscontrollers/default -o yaml
```

- The **Progressing** status condition indicates whether you must take further action. For example, the status condition can indicate that you need to delete the service by entering the following command:

```
$ oc -n openshift-ingress delete services/router-default
```

If you delete the service, the Ingress Operator recreates it as **External**.

2.4.1.3. Adding a single NodePort service to an Ingress Controller

To prevent port conflicts, instead of creating a **NodePort**-type **Service** for each project, create a custom Ingress Controller that can use the **NodePortService** endpoint publishing strategy.

Consider this configuration for your Ingress Controller when you want to apply a set of routes, through Ingress sharding, to nodes that might already have a **HostNetwork** Ingress Controller.

Before you set a **NodePort**-type **Service** for each project, read the following considerations:

- You must create a wildcard DNS record for the **Nodeport** Ingress Controller domain. A Nodeport Ingress Controller route can be reached from the address of a worker node. For more information about the required DNS records for routes, see "User-provisioned DNS requirements".
- You must expose a route for your service and specify the **--hostname** argument for your custom Ingress Controller domain.
- You must append the port that is assigned to the **NodePort**-type **Service** in the route so that you can access application pods.

Prerequisites

- You installed the OpenShift CLI (**oc**).
- Logged in as a user with **cluster-admin** privileges.
- You created a wildcard DNS record.

Procedure

1. Create a custom resource (CR) file for the Ingress Controller:

   Example of a CR file that defines information for the IngressController object

```
apiVersion: v1
items:
- apiVersion: operator.openshift.io/v1
  kind: IngressController
  metadata:
    name: <custom_ic_name>
    namespace: openshift-ingress-operator
  spec:
    replicas: 1
    domain: <custom_ic_domain_name>
    nodePlacement:
      nodeSelector:
        matchLabels:
          <key>: <value>
    namespaceSelector:
     matchLabels:
       <key>: <value>
```

```
    endpointPublishingStrategy:
      type: NodePortService
  # ...
```

where:

metadata.name

Specifies a custom **name** for the **IngressController** CR.

spec.domain

Specifies the DNS name that the Ingress Controller services. For example, the default ingresscontroller domain is **apps.ipi-cluster.example.com**, so you would specify the **<custom_ic_domain_name>** as **nodeportsvc.ipi-cluster.example.com**.

nodeSelector.matchLabels.<key>

Specifies the label for the nodes that include the custom Ingress Controller.

namespaceSelector.matchLabels.<key>

Specifies the label for a set of namespaces. Substitute **<key>:<value>** with a map of key-value pairs where **<key>** is a unique name for the new label and **<value>** is its value. For example: **ingresscontroller: custom-ic**.

2. Add a label to a node by using the **oc label node** command:

```
$ oc label node <node_name> <key>=<value>
```

- **<key>=<value>**: Where **<value>** must match the key-value pair specified in the **nodePlacement** section of your **IngressController** CR.

3. Create the **IngressController** object:

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

4. Find the port for the service created for the **IngressController** CR:

```
$ oc get svc -n openshift-ingress
```

Example output that shows port 80:32432/TCP for the router-nodeport-custom-ic3 service

```
NAME                        TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)
AGE
router-internal-default     ClusterIP   172.30.195.74    <none>
80/TCP,443/TCP,1936/TCP                 223d
router-nodeport-custom-ic3  NodePort    172.30.109.219   <none>
80:32432/TCP,443:31366/TCP,1936:30499/TCP   155m
```

5. To create a new project, enter the following command:

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

6. To label the new namespace, enter the following command:

```
$ oc label namespace <project_name> <key>=<value>
```

- **<key>=<value>**:: Where **<key>=<value>** must match the value in the **namespaceSelector** section of your Ingress Controller CR.

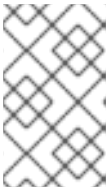
7. Create a new application in your cluster:

```
$ oc new-app --image=<image_name>
```

- **<image_name>**: An example of **<image_name>** is **quay.io/openshifttest/hello-openshift:multiarch**.

8. Create a **Route** object for a service, so that the pod can use the service to expose the application external to the cluster.

```
$ oc expose svc/<service_name> --hostname=<svc_name>-<project_name>.
<custom_ic_domain_name>
```



NOTE

You must specify the domain name of your custom Ingress Controller in the **--hostname** argument. If you do not do this, the Ingress Operator uses the default Ingress Controller to serve all the routes for your cluster.

9. Check that the route has the **Admitted** status and that it includes metadata for the custom Ingress Controller:

```
$ oc get route/hello-openshift -o json | jq '.status.ingress'
```

Example output

```
# ...
{
  "conditions": [
    {
      "lastTransitionTime": "2024-05-17T18:25:41Z",
      "status": "True",
      "type": "Admitted"
    }
  ],
  [
    {
      "host": "hello-openshift.nodeportsvc.ipi-cluster.example.com",
      "routerCanonicalHostname": "router-nodeportsvc.nodeportsvc.ipi-cluster.example.com",
      "routerName": "nodeportsvc", "wildcardPolicy": "None"
    }
  ],
}
```

10. Update the default **IngressController** CR to prevent the default Ingress Controller from managing the **NodePort**-type **Service**. The default Ingress Controller will continue to monitor all other cluster traffic.

```
$ oc patch --type=merge -n openshift-ingress-operator ingresscontroller/default --patch
'{"spec":{"namespaceSelector":{"matchExpressions":[{"key":"
<key>","operator":"NotIn","values":["<value>]}]}}}'
```

Verification

1. Verify that the DNS entry can route inside and outside of your cluster by entering the following command. The command outputs the IP address of the node that received the label from running the **oc label node** command earlier in the procedure.

```
$ dig +short <svc_name>-<project_name>.<custom_ic_domain_name>
```

2. To verify that your cluster uses the IP addresses from external DNS servers for DNS resolution, check the connection of your cluster by entering the following command:

```
$ curl <svc_name>-<project_name>.<custom_ic_domain_name>:<port>
```
(1)

- **<custom_ic_domain_name>:<port>**: Where **<port>** is the node port from the **NodePort**-type **Service**. Based on example output from the **oc get svc -n openshift-ingress** command, the **80:32432/TCP** HTTP route means that **32432** is the node port.

  Output example

```
Hello OpenShift!
```

2.4.2. Additional resources

- Ingress Controller configuration parameters
- Setting RHOSP Cloud Controller Manager options
- User-provisioned DNS requirements

2.5. CONFIGURING INGRESS CLUSTER TRAFFIC USING A LOAD BALANCER

OpenShift Container Platform provides methods for communicating from outside the cluster with services running in the cluster. This method uses a load balancer.

Before starting the following procedures, the administrator must complete the following prerequisite tasks:

- Set up the external port to the cluster networking environment so that requests can reach the cluster.
- Have an OpenShift Container Platform cluster with at least one control plane node, at least one compute node, and a system outside the cluster that has network access to the cluster. This procedure assumes that the external system is on the same subnet as the cluster. The additional networking required for external systems on a different subnet is out-of-scope for this topic.
- Make sure there is at least one user with cluster admin role. To add this role to a user, run the following command:

```
$ oc adm policy add-cluster-role-to-user cluster-admin username
```

2.5.1. Using a load balancer to get traffic into the cluster

If you do not need a specific external IP address, you can configure a load balancer service to allow external access to an OpenShift Container Platform cluster.

A load balancer service allocates a unique IP. The load balancer has a single edge router IP, which can be a virtual IP (VIP), but is still a single machine for initial load balancing.

NOTE

A pool gets configured at the infrastructure level and not the cluster administrator level.

2.5.2. Creating a project and service

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

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

Prerequisites

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

Procedure

1. Create a new project for your service by running the **oc new-project** command:

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

2. Use the **oc new-app** command to create your service:

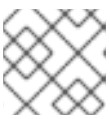
```
$ oc new-app nodejs:12~https://github.com/sclorg/nodejs-ex.git
```

3. To verify that the service was created, run the following command:

```
$ oc get svc -n <project_name>
```

Example output

```
NAME       TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
nodejs-ex  ClusterIP  172.30.197.157  <none>        8080/TCP   70s
```



NOTE

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

2.5.3. Exposing the service by creating a route

To enable external access to your application that runs on OpenShift Container Platform, you can expose the service as a route by using the **oc expose** command.

Prerequisites

- You logged into OpenShift Container Platform.

Procedure

1. Log in to the project where the service you want to expose is located:

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

2. Run the **oc expose service** command to expose the route:

```
$ oc expose service nodejs-ex
```

Example output

```
route.route.openshift.io/nodejs-ex exposed
```

3. To verify that the service is exposed, you can use a tool, such as **curl** to check that the service is accessible from outside the cluster.

   a. To find the hostname of the route, enter the following command:

```
$ oc get route
```

   Example output

```
NAME       HOST/PORT                         PATH   SERVICES   PORT       TERMINATION   WILDCARD
nodejs-ex  nodejs-ex-myproject.example.com          nodejs-ex  8080-tcp                 None
```

   b. To check that the host responds to a GET request, enter the following command:

   Example curl command

```
$ curl --head nodejs-ex-myproject.example.com
```

   Example output

```
HTTP/1.1 200 OK
...
```

2.5.4. Creating a load balancer service

To distribute incoming traffic efficiently and ensure high availability for your applications in OpenShift Container Platform, create a load balancer service.

Prerequisites

- Make sure that the project and service you want to expose exist.
- Your cloud provider supports load balancers.

Procedure

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

   Example command

   ```
   $ oc project project1
   ```

3. Open a text file on the control plane node and paste the following text into the file. Edit the file as needed.

   Sample load balancer configuration file

   ```
   apiVersion: v1
   kind: Service
   metadata:
     name: egress-2
   spec:
     ports:
     - name: db
       port: 3306
     loadBalancerIP:
     loadBalancerSourceRanges:
     - 10.0.0.0/8
     - 192.168.0.0/16
     type: LoadBalancer
     selector:
       name: mysql
   ```

   where:

   metadata.name

   Specifies a descriptive name for the load balancer service.

   ports.port

   Specifies the same port that the service you want to expose is listening on.

   loadBalancerSourceRanges

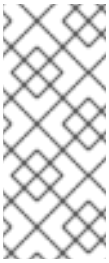
   Specifies a list of specific IP addresses to restrict traffic through the load balancer. The parameter is ignored if the cloud provider does not support the feature.

   type

   Specifies **Loadbalancer** as the type.

   selector.name

   Specifies the name of the service.



NOTE

To restrict the traffic through the load balancer to specific IP addresses, use the **spec.endpointPublishingStrategy.loadBalancer.allowedSourceRanges** Ingress Controller parameter. Do not set the **loadBalancerSourceRanges** parameter.

4. Save and exit the file.

5. Run the following command to create the service:

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

For example:

```
$ oc create -f mysql-lb.yaml
```

6. Execute the following command to view the new service:

```
$ oc get svc
```

Example output

```
NAME      TYPE          CLUSTER-IP      EXTERNAL-IP                          PORT(S)
AGE
egress-2  LoadBalancer  172.30.22.226   ad42f5d8b303045-487804948.example.com
3306:30357/TCP  15m
```

The service has an external IP address automatically assigned if there is a cloud provider enabled.

7. On the master, use a tool, such as **curl**, to make sure you can reach the service by using the public IP address:

```
$ curl <public_ip>:<port>
```

For example:

```
$ curl 172.29.121.74:3306
```

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

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

```
$ mysql -h 172.30.131.89 -u admin -p
```

Example output

```
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.

MySQL [(none)]>
```

2.6. CONFIGURING INGRESS CLUSTER TRAFFIC ON AWS

OpenShift Container Platform provides methods for communicating from outside the cluster with services running in the cluster. This method uses load balancers on Amazon Web Services (AWS), specifically a Network Load Balancer (NLB) or a Classic Load Balancer (CLB). Both types of load balancers can forward the IP address of the client to the node, but a CLB requires proxy protocol support, which OpenShift Container Platform automatically enables.

There are two ways to configure an Ingress Controller to use an NLB:

1. By force replacing the Ingress Controller that is currently using a CLB. This deletes the **IngressController** object and an outage occurs while the new DNS records propagate and the NLB is being provisioned.

2. By editing an existing Ingress Controller that uses a CLB to then use an NLB. This changes the load balancer without having to delete and recreate the **IngressController** object.

Both methods can be used to switch from an NLB to a CLB.

You can configure these load balancers on a new or existing AWS cluster.

2.6.1. Configuring Classic Load Balancer timeouts on AWS

To prevent connection drops for long-running processes in OpenShift Container Platform, configure custom timeout periods for specific routes or Ingress Controllers.

Ensure these settings account for the Amazon Web Services Classic Load Balancer (CLB) default timeout of 60 seconds to maintain stable network traffic.

If the timeout period of the CLB is shorter than the route timeout or Ingress Controller timeout, the load balancer can prematurely terminate the connection. You can prevent this problem by increasing both the timeout period of the route and CLB.

2.6.1.1. Configuring route timeouts

You can configure the default timeouts for an existing route when you have services in need of a low timeout, which is required for Service Level Availability (SLA) purposes, or a high timeout, for cases with a slow back end.

IMPORTANT

If you configured a user-managed external load balancer in front of your OpenShift Container Platform cluster, ensure that the timeout value for the user-managed external load balancer is higher than the timeout value for the route. This configuration prevents network congestion issues over the network that your cluster uses.

Prerequisites

- You deployed an Ingress Controller on a running cluster.

Procedure

1. Using the **oc annotate** command, add the timeout to the route:

```
$ oc annotate route <route_name> \
    --overwrite haproxy.router.openshift.io/timeout=<timeout><time_unit>
```

- **<timeout>**: Supported time units are microseconds (us), milliseconds (ms), seconds (s), minutes (m), hours (h), or days (d).
  The following example sets a timeout of two seconds on a route named **myroute**:

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

2.6.1.2. Configuring Classic Load Balancer timeouts

You can configure the default timeouts for a Classic Load Balancer (CLB) to extend idle connections.

Prerequisites

- You must have a deployed Ingress Controller on a running cluster.

Procedure

1. Set an Amazon Web Services connection idle timeout of five minutes for the default **ingresscontroller** by running the following command:

```
$ oc -n openshift-ingress-operator patch ingresscontroller/default \
    --type=merge --patch='{"spec":{"endpointPublishingStrategy": \
    {"type":"LoadBalancerService", "loadBalancer": \
    {"scope":"External", "providerParameters":{"type":"AWS", "aws": \
    {"type":"Classic", "classicLoadBalancer": \
    {"connectionIdleTimeout":"5m"}}}}}}'
```

2. Optional: Restore the default value of the timeout by running the following command:

```
$ oc -n openshift-ingress-operator patch ingresscontroller/default \
    --type=merge --patch='{"spec":{"endpointPublishingStrategy": \
    {"loadBalancer":{"providerParameters":{"aws":{"classicLoadBalancer": \
    {"connectionIdleTimeout":null}}}}}}'
```

NOTE

You must specify the **scope** field when you change the connection timeout value unless the current scope is already set. When you set the **scope** field, you do not need to do so again if you restore the default timeout value.

2.6.2. Configuring ingress cluster traffic on AWS using a Network Load Balancer

To enable high-performance communication between external services and your OpenShift Container Platform cluster, configure an Amazon Web Services Network Load Balancer (NLB). You can set up an NLB on a new or existing AWS cluster to manage ingress traffic with low latency.

2.6.2.1. Switching the Ingress Controller from using a Classic Load Balancer to a Network Load Balancer

To improve performance and reduce latency for cluster traffic in OpenShift Container Platform on Amazon Web Services, switch an Ingress Controller using a Classic Load Balancer (CLB) to one that uses a Network Load Balancer (NLB).

Switching between these load balancers does not delete the **IngressController** object.

WARNING

This procedure might cause the following issues:

- An outage that can last several minutes due to new DNS records propagation, new load balancers provisioning, and other factors. IP addresses and canonical names of the Ingress Controller load balancer might change after applying this procedure.
- Leaked load balancer resources due to a change in the annotation of the service.

Procedure

1. Modify the existing Ingress Controller that you want to switch to by using an NLB. This example assumes that your default Ingress Controller has an **External** scope and no other customizations:

   Example ingresscontroller.yaml file

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  creationTimestamp: null
  name: default
  namespace: openshift-ingress-operator
spec:
  endpointPublishingStrategy:
    loadBalancer:
      scope: External
      providerParameters:
        type: AWS
        aws:
          type: NLB
    type: LoadBalancerService
```

NOTE

If you do not specify a value for the **spec.endpointPublishingStrategy.loadBalancer.providerParameters.aws.type** field, the Ingress Controller uses the **spec.loadBalancer.platform.aws.type** value from the cluster **Ingress** configuration that was set during installation.

TIP

If your Ingress Controller has other customizations that you want to update, such as changing the domain, consider force replacing the Ingress Controller definition file instead.

2. Apply the changes to the Ingress Controller YAML file by running the command:

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

Expect several minutes of outages while the Ingress Controller updates.

2.6.2.2. Switching the Ingress Controller from using a Network Load Balancer to a Classic Load Balancer

To support specific networking configurations in OpenShift Container Platform on Amazon Web Services, switch an Ingress Controller using a Network Load Balancer (NLB) to one that uses a Classic Load Balancer (CLB).

Switching between these load balancers does not delete the **IngressController** object.

WARNING

This procedure might cause an outage that can last several minutes due to new DNS records propagation, new load balancers provisioning, and other factors. IP addresses and canonical names of the Ingress Controller load balancer might change after applying this procedure.

Procedure

1. Modify the existing Ingress Controller that you want to switch to using a CLB. This example assumes that your default Ingress Controller has an **External** scope and no other customizations:

Example ingresscontroller.yaml file

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  creationTimestamp: null
  name: default
  namespace: openshift-ingress-operator
spec:
```

```
endpointPublishingStrategy:
  loadBalancer:
    scope: External
    providerParameters:
      type: AWS
      aws:
        type: Classic
  type: LoadBalancerService
```

NOTE

If you do not specify a value for the **spec.endpointPublishingStrategy.loadBalancer.providerParameters.aws.type** field, the Ingress Controller uses the **spec.loadBalancer.platform.aws.type** value from the cluster **Ingress** configuration that was set during installation.

TIP

If your Ingress Controller has other customizations that you want to update, such as changing the domain, consider force replacing the Ingress Controller definition file instead.

2. Apply the changes to the Ingress Controller YAML file by running the command:

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

Expect several minutes of outages while the Ingress Controller updates.

2.6.2.3. Replacing Ingress Controller Classic Load Balancer with Network Load Balancer

To improve performance and reduce latency for traffic in OpenShift Container Platform on Amazon Web Services, replace an Ingress Controller using a Classic Load Balancer (CLB) with one that uses a Network Load Balancer (NLB).

WARNING

This procedure might cause the following issues:

- An outage that can last several minutes due to new DNS records propagation, new load balancers provisioning, and other factors. IP addresses and canonical names of the Ingress Controller load balancer might change after applying this procedure.
- Leaked load balancer resources due to a change in the annotation of the service.

Procedure

1. Create a file with a new default Ingress Controller. The following example assumes that your default Ingress Controller has an **External** scope and no other customizations:

Example ingresscontroller.yml file

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  creationTimestamp: null
  name: default
  namespace: openshift-ingress-operator
spec:
  endpointPublishingStrategy:
    loadBalancer:
      scope: External
      providerParameters:
        type: AWS
        aws:
          type: NLB
    type: LoadBalancerService
```

If your default Ingress Controller has other customizations, ensure that you modify the file accordingly.

TIP

If your Ingress Controller has no other customizations and you are only updating the load balancer type, consider following the procedure detailed in "Switching the Ingress Controller from using a Classic Load Balancer to a Network Load Balancer".

2. Force replace the Ingress Controller YAML file:

```
$ oc replace --force --wait -f ingresscontroller.yml
```

Wait until the Ingress Controller is replaced. Expect several of minutes of outages.

2.6.2.4. Configuring an Ingress Controller Network Load Balancer on an existing AWS cluster

To improve performance for high-traffic workloads in OpenShift Container Platform, configure an Ingress Controller backed by an Amazon Web Services Network Load Balancer (NLB) on an existing cluster.

You can create an Ingress Controller backed by an Amazon Web Services Network Load Balancer (NLB) on an existing cluster.

Prerequisites

- You installed an AWS cluster.
- **PlatformStatus** of the infrastructure resource must be AWS.
  - To verify that the **PlatformStatus** is AWS, run the following command:

```
$ oc get infrastructure/cluster -o jsonpath='{.status.platformStatus.type}'
AWS
```

Procedure

1. Create the Ingress Controller manifest:

```
$ cat ingresscontroller-aws-nlb.yaml
```

Example output

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: <ingress_controller_name>
  namespace: openshift-ingress-operator
spec:
  domain: <unique_ingress_domain
  endpointPublishingStrategy:
    type: LoadBalancerService
    loadBalancer:
      scope: External
      providerParameters:
        type: AWS
        aws:
          type: NLB
```

where:

<ingress_controller_name>

Specifies a unique name for the Ingress Controller.

<unique_ingress_domain>

Specifies a domain name that is unique among all Ingress Controllers in the cluster. This variable must be a subdomain of the DNS name **<clustername>.<domain>**.

scope

Specifies the type of NLB, either **External** to use an external NLB or **Internal** to use an internal NLB.

2. Create the resource in the cluster:

```
$ oc create -f ingresscontroller-aws-nlb.yaml
```

IMPORTANT

Before you can configure an Ingress Controller NLB on a new AWS cluster, you must complete the creating the installation configuration file procedure. For more information, see "Creating the installation configuration file".

2.6.3. Additional resources

- Creating the installation configuration file

2.6.3.1. Configuring an Ingress Controller Network Load Balancer on a new AWS cluster

You can create an Ingress Controller backed by an Amazon Web Services Network Load Balancer (NLB) on a new cluster in situations where you need more transparent networking capabilities.

Prerequisites

- Create and edit the **install-config.yaml** file. For instructions, see "Creating the installation configuration file" in the *Additonal resources* section.

Procedure

1. Change to the directory that contains the installation program and create the manifests:

```
$ ./openshift-install create manifests --dir <installation_directory>
```

   - For **<installation_directory>**, specify the name of the directory that contains the **install-config.yaml** file for your cluster.

2. Create a file that is named **cluster-ingress-default-ingresscontroller.yaml** in the **<installation_directory>/manifests/** directory:

```
$ touch <installation_directory>/manifests/cluster-ingress-default-ingresscontroller.yaml
```

   <installation_directory>

   Specifies the directory name that contains the **manifests/** directory for your cluster.

3. Check the several network configuration files that exist in the **manifests/** directory by entering the following command:

```
$ ls <installation_directory>/manifests/cluster-ingress-default-ingresscontroller.yaml
```

   #### Example output

```
cluster-ingress-default-ingresscontroller.yaml
```

4. Open the **cluster-ingress-default-ingresscontroller.yaml** file in an editor and enter a custom resource (CR) that describes the Operator configuration you want:

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  creationTimestamp: null
  name: default
  namespace: openshift-ingress-operator
spec:
  endpointPublishingStrategy:
    loadBalancer:
      scope: External
      providerParameters:
        type: AWS
        aws:
          type: NLB
    type: LoadBalancerService
```

5. Save the **cluster-ingress-default-ingresscontroller.yaml** file and quit the text editor.

6. Optional: Back up the **manifests/cluster-ingress-default-ingresscontroller.yaml** file because the installation program deletes the **manifests/** directory during cluster creation.

2.6.3.2. Choosing subnets while creating a LoadBalancerService Ingress Controller

To manually control network placement for Ingress Controllers in an existing cluster, specify the load balancer subnets in your configuration. This method provides precise control over your infrastructure by overriding the default automatic subnet discovery method used by Amazon Web Services.

Prerequisites

- You must have an installed AWS cluster.
- You must know the names or IDs of the subnets to which you intend to map your **IngressController**.

Procedure

1. Create a custom resource (CR) YAML file, such as **sample-ingress.yaml**, and specifying the following content for the file:

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  namespace: openshift-ingress-operator
  name: <name>
spec:
  domain: <domain>
  endpointPublishingStrategy:
    type: LoadBalancerService
    loadBalancer:
      scope: External
  dnsManagementPolicy: Managed
# ...
```

2. Add subnets to the CR file:

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name:  <name>
  namespace: openshift-ingress-operator
spec:
  domain: <domain>
  endpointPublishingStrategy:
    type: LoadBalancerService
    loadBalancer:
      scope: External
      providerParameters:
        type: AWS
        aws:
          type: Classic
          classicLoadBalancer:
```

```
            subnets:
              ids:
              - <subnet>
              - <subnet>
              - <subnet>
dnsManagementPolicy: Managed
```

where:

name

Specifies a name for the **IngressController**.

domain

Specifies the DNS name serviced by the **IngressController**.

classicLoadBalancer

Specifies the type of load balancer, either **classicLoadBalancer** if using a CLB or **networkLoadBalancer** field if using an NLB.

ids

Specifies a subnet by name using the **names** field instead of specifying the subnet by ID. This field is optional.

<subnet>

Specifies the subnet IDs (or names if you using **names**).

IMPORTANT

You can specify a maximum of one subnet per availability zone. Only provide public subnets for external Ingress Controllers and private subnets for internal Ingress Controllers.

3. Save and apply the CR file by using the OpenShift CLI (**oc**):

```
$  oc apply -f sample-ingress.yaml
```

4. Confirm the load balancer was provisioned successfully by checking the **IngressController** conditions.

```
$ oc get ingresscontroller -n openshift-ingress-operator <name> -o jsonpath="
{.status.conditions}" | yq -PC
```

2.6.3.3. Updating the subnets on an existing Ingress Controller

You can update an **IngressController** with manually specified load balancer subnets in OpenShift Container Platform to avoid any disruptions, to maintain the stability of your services, and to ensure that your network configuration aligns with your specific requirements.

The example in the procedure shows you how to select and apply new subnets, verify the configuration changes, and confirm successful load balancer provisioning.

WARNING

This procedure may cause an outage that can last several minutes due to new DNS records propagation, new load balancers provisioning, and other factors. IP addresses and canonical names of the Ingress Controller load balancer might change after applying this procedure.

Procedure

1. Modify the existing IngressController by specifying the new subnets:

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name:  <name>
  namespace: openshift-ingress-operator
spec:
  domain: <domain>
  endpointPublishingStrategy:
    type: LoadBalancerService
    loadBalancer:
      scope: External
      providerParameters:
        type: AWS
        aws:
          type: Classic
          classicLoadBalancer:
            subnets:
              ids:
              - <updated_subnet>
              - <updated_subnet>
              - <updated_subnet>
# ...
```

where:

<name>

Specifies a name for the **IngressController**.

<domain>

Specifies the DNS name serviced by the **IngressController**.

type

Specifies the updated subnet IDs (or names if you using **names**).

classicLoadBalancer

You can also use the **networkLoadBalancer** field if using an NLB.

ids

Specifies the subnet by name using the **names** field instead of specifying the subnet by ID. This parameter is optional.

<updated_subnet>

Specifies the updated subnet IDs (or names if you are using **names**).

IMPORTANT

You can specify a maximum of one subnet per availability zone. Only provide public subnets for external Ingress Controllers and private subnets for internal Ingress Controllers.

2. Examine the **Progressing** condition on the **IngressController** for instructions on how to apply the subnet updates by running the following command:

```
$ oc get ingresscontroller -n openshift-ingress-operator subnets -o jsonpath="
{.status.conditions[?(@.type==\"Progressing\")]}" | yq -PC
```

Example output

```
lastTransitionTime: "2024-11-25T20:19:31Z"
message: 'One or more status conditions indicate progressing:
LoadBalancerProgressing=True (OperandsProgressing: One or more managed resources
are progressing: The IngressController subnets were changed from [...] to [...].  To effectuate
this change, you must delete the service: `oc -n openshift-ingress delete svc/router-<name>`;
the service load-balancer will then be deprovisioned and a new one created. This will most
likely cause the new load-balancer to have a different host name and IP address and cause
disruption. To return to the previous state, you can revert the change to the
IngressController: [...]'
reason: IngressControllerProgressing
status: "True"
type: Progressing
```

3. To apply the update, delete the service associated with the Ingress controller by running the following command:

```
$ oc -n openshift-ingress delete svc/router-<name>
```

Verification

- To confirm that the load balancer was provisioned successfully, check the **IngressController** conditions by running the following command:

```
$ oc get ingresscontroller -n openshift-ingress-operator <name> -o jsonpath="
{.status.conditions}" | yq -PC
```

2.6.3.4. Configuring AWS Elastic IP (EIP) addresses for a Network Load Balancer (NLB)

You can specify static IPs, otherwise known as elastic IPs, for your network load balancer (NLB) in the Ingress Controller. This is useful in situations where you want to configure appropriate firewall rules for your cluster network.

Prerequisites

- You must have an installed Amazon Web Services cluster.
- You must know the names or IDs of the subnets to which you intend to map your **IngressController**.

Procedure

1. Create a YAML file that contains the following example content:

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  namespace: openshift-ingress-operator
  name: <name>
spec:
  domain: <domain>
  endpointPublishingStrategy:
    loadBalancer:
      scope: External
      type: LoadBalancerService
      providerParameters:
        type: AWS
        aws:
          type: NLB
          networkLoadBalancer:
            subnets:
              ids:
              - <subnet_ID>
              names:
              - <subnet_A>
              - <subnet_B>
            eipAllocations:
            - <eipalloc_A>
            - <eipalloc_B>
            - <eipalloc_C>
```

where:

<name>

Specifies a name for the Ingress Controller.

<domain>

Specifies the DNS name serviced by the Ingress Controller.

scope

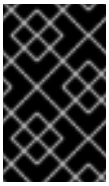
Specifies a scope for the EIPs. The scope must be set to the value **External** and be Internet-facing in order to allocate EIPs.

`subnets

Specifies the IDs and names for your subnets. The total number of IDs and names must be equal to your allocated EIPs.

eipAllocations

Specifies the EIP addresses.



IMPORTANT

You can specify a maximum of one subnet per availability zone. Only provide public subnets for external Ingress Controllers. You can associate one EIP address per subnet.

2. Save and apply the CR file by entering the following command:

```
$  oc apply -f sample-ingress.yaml
```

Verification

1. Confirm the load balancer was provisioned successfully by checking the **IngressController** conditions by running the following command:

```
$ oc get ingresscontroller -n openshift-ingress-operator <name> -o jsonpath="
{.status.conditions}" | yq -PC
```

2.6.4. Additional resources

- Installing a cluster on AWS with network customizations
- Network Load Balancer support on AWS
- Configure proxy protocol support for your Classic Load Balancer

2.7. CONFIGURING INGRESS CLUSTER TRAFFIC FOR A SERVICE EXTERNAL IP

You can use either a MetalLB implementation or an IP failover deployment to attach an ExternalIP resource to a service so that the service is available to traffic outside your OpenShift Container Platform cluster.

Hosting an external IP address in this way is only applicable for a cluster installed on bare-metal hardware.

You must ensure that you correctly configure the external network infrastructure to route traffic to the service.

Before you begin the procedure, ensure that you meet the following prerequisite:

- You configured your cluster with ExternalIPs enabled. For more information, see "Configuring ExternalIPs for services" in the *Additional resources* section.

NOTE

Do not use the same ExternalIP for the egress IP.

2.7.1. Attaching an ExternalIP to a service

To route traffic from external networks to your applications in OpenShift Container Platform, attach an ExternalIP resource to a service.

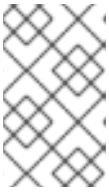
If you configured your cluster to automatically attach the resource to a service, you might not need to manually attach an ExternalIP to the service.

The examples in the procedure use a scenario that manually attaches an ExternalIP resource to a service in a cluster with an IP failover configuration.

Procedure

1. Confirm compatible IP address ranges for the ExternalIP resource by entering the following command in your CLI:

```
$ oc get networks.config cluster -o jsonpath='{.spec.externalIP}{"\n"}'
```



NOTE

If **autoAssignCIDRs** is set and you did not specify a value for **spec.externalIPs** in the ExternalIP resource, OpenShift Container Platform automatically assigns ExternalIP to a new **Service** object.

2. Choose one of the following options to attach an ExternalIP resource to the service:

   a. If you are creating a new service, specify a value in the **spec.externalIPs** parameter and array of one or more valid IP addresses in the **allowedCIDRs** parameter.

   Example of service YAML configuration file that supports an ExternalIP resource

```
apiVersion: v1
kind: Service
metadata:
  name: svc-with-externalip
spec:
  externalIPs:
    policy:
      allowedCIDRs:
      - 192.168.123.0/28
# ...
```

   b. If you are attaching an ExternalIP to an existing service, enter the following command. Replace **<name>** with the service name. Replace **<ip_address>** with a valid ExternalIP address. You can provide multiple IP addresses separated by commas.

```
$ oc patch svc <name> -p \
  '{
    "spec": {
      "externalIPs": [ "<ip_address>" ]
    }
  }'
```

   For example:

```
$ oc patch svc mysql-55-rhel7 -p '{"spec":{"externalIPs":["192.174.120.10"]}}'
```

Example output

```
"mysql-55-rhel7" patched
```

3. To confirm that an ExternalIP address is attached to the service, enter the following command. If you specified an ExternalIP for a new service, you must create the service first.

```
$ oc get svc
```

Example output

```
NAME            CLUSTER-IP      EXTERNAL-IP     PORT(S)    AGE
mysql-55-rhel7    172.30.131.89   192.174.120.10  3306/TCP   13m
```

2.7.2. Additional resources

Configuring ExternalIPs for services

- About MetalLB and the MetalLB Operator
- Configuring IP failover
- Configuring ExternalIPs for services

2.8. CONFIGURING INGRESS CLUSTER TRAFFIC BY USING A NODEPORT

To enable external access to your application for specific networking requirements, expose a service by using a **NodePort**.

This configuration opens a specific port on every node in the cluster, allowing external traffic to reach your workloads by using an IP address of any node.

OpenShift Container Platform provides methods for communicating from outside the cluster with services running in the cluster. This method uses a **NodePort**.

Before starting the following procedures, the administrator must complete the following prerequisite tasks:

- Set up the external port to the cluster networking environment so that requests can reach the cluster.
- Have an OpenShift Container Platform cluster with at least one control plane node, at least one compute node, and a system outside the cluster that has network access to the cluster. This procedure assumes that the external system is on the same subnet as the cluster. The additional networking required for external systems on a different subnet is out-of-scope for this topic.
- Make sure there is at least one user with cluster admin role. To add this role to a user, run the following command:

```
$ oc adm policy add-cluster-role-to-user cluster-admin <user_name>
```

2.8.1. Using a NodePort to get traffic into the cluster

Use a **NodePort**-type **Service** resource to expose a service on a specific port on all nodes in the cluster.

The port is specified in the **Service** resource's **.spec.ports[*].nodePort** parameter

IMPORTANT

Using a node port requires additional port resources.

A **NodePort** exposes the service on a static port on the IP address of a node. A **NodePort** spans the **30000** to **32767** IP address ranges by default, which means a **NodePort** is unlikely to match the intended port of a service. For example, port **8080** might be exposed as port **31020** on the node.

The administrator must ensure the external IP addresses are routed to the nodes.

A **NodePort** and external IPs are independent and both can be used concurrently.

2.8.2. Creating a project and service

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

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

Prerequisites

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

Procedure

1. Create a new project for your service by running the **oc new-project** command:

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

2. Use the **oc new-app** command to create your service:

```
$ oc new-app nodejs:12~https://github.com/sclorg/nodejs-ex.git
```

3. To verify that the service was created, run the following command:

```
$ oc get svc -n <project_name>
```

Example output

```
NAME       TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
nodejs-ex  ClusterIP  172.30.197.157  <none>        8080/TCP   70s
```

NOTE

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

2.8.3. Exposing the service by creating a route

To enable external access to your application that runs on OpenShift Container Platform, you can expose the service as a route by using the **oc expose** command.

Prerequisites

- You logged into OpenShift Container Platform.

Procedure

1. Log in to the project where the service you want to expose is located:

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

2. To expose a node port for the application, modify the custom resource definition (CRD) of a service by entering the following command:

```
$ oc edit svc <service_name>
```

Example output

```
spec:
  ports:
  - name: 8443-tcp
    nodePort: 30327
    port: 8443
    protocol: TCP
    targetPort: 8443
  sessionAffinity: None
  type: NodePort
```

- **nodePort**: Optional parameter. Specifies the node port range for the application. By default, OpenShift Container Platform selects an available port in the **30000-32767** range.
- **type**: Specifies the service type.

3. Optional: To confirm the service is available with a node port exposed, enter the following command:

```
$ oc get svc -n myproject
```

Example output

```
NAME              TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
nodejs-ex         ClusterIP   172.30.217.127  <none>        3306/TCP         9m44s
nodejs-ex-ingress NodePort    172.30.107.72   <none>        3306:31345/TCP   39s
```

4. Optional: To remove the service created automatically by the **oc new-app** command, enter the following command:

```
$ oc delete svc nodejs-ex
```

Verification

- To check that the service node port is updated with a port in the **30000-32767** range, enter the following command:

```
$ oc get svc
```

In the following example output, the updated port is **30327**:

Example output

```
NAME    TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
httpd   NodePort   172.xx.xx.xx    <none>        8443:30327/TCP   109s
```

2.8.4. Additional resources

- Configuring the node port service range
- Adding a single NodePort service to an Ingress Controller

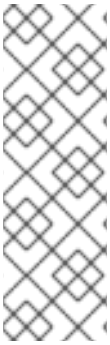
2.9. CONFIGURING INGRESS CLUSTER TRAFFIC USING LOAD BALANCER ALLOWED SOURCE RANGES

You can specify a list of IP address ranges for the Ingress Controller. This action restricts access to the load balancer service when you specify the **LoadBalancerService** value for the **endpointPublishingStrategy** parameter.

2.9.1. Configuring load balancer allowed source ranges

You can enable and configure the **spec.endpointPublishingStrategy.loadBalancer.allowedSourceRanges** parameter. By configuring load balancer allowed source ranges, you can limit the access to the load balancer for the Ingress Controller to a specified list of IP address ranges.

The Ingress Operator reconciles the load balancer Service and sets the **spec.loadBalancerSourceRanges** parameter based on **AllowedSourceRanges**.



NOTE

If you have already set the **spec.loadBalancerSourceRanges** parameter or the load balancer service anotation **service.beta.kubernetes.io/load-balancer-source-ranges** in a previous version of OpenShift Container Platform, Ingress Controller starts reporting **Progressing=True** after an upgrade. To fix this, set **AllowedSourceRanges** that overwrites the **spec.loadBalancerSourceRanges** parameter and clears the **service.beta.kubernetes.io/load-balancer-source-ranges** annotation. Ingress Controller starts reporting **Progressing=False** again.

Prerequisites

- You have a deployed Ingress Controller on a running cluster.

Procedure

- Set the allowed source ranges API for the Ingress Controller by running the following command:

```
$ oc -n openshift-ingress-operator patch ingresscontroller/default \
    --type=merge --patch='{"spec":{"endpointPublishingStrategy": \
    {"type":"LoadBalancerService", "loadbalancer": \
    {"scope":"External", "allowedSourceRanges":["0.0.0.0/0"]}}}}'
```

where:

allowedSourceRanges

The example value **0.0.0.0/0** specifies the allowed source range.

2.9.2. Migrating to load balancer allowed source ranges

To ensure long-term compatibility and use stable API parameters in OpenShift Container Platform, migrate from the legacy **service.beta.kubernetes.io/load-balancer-source-ranges** annotation to load balancer allowed source ranges.

When you set the **AllowedSourceRanges**, the Ingress Controller sets the **spec.loadBalancerSourceRanges** parameter based on the **AllowedSourceRanges** value and unsets the **service.beta.kubernetes.io/load-balancer-source-ranges** annotation.

NOTE

If you have already set the **spec.loadBalancerSourceRanges** parameter or the load balancer service anotation **service.beta.kubernetes.io/load-balancer-source-ranges** in a previous version of OpenShift Container Platform, the Ingress Controller starts reporting **Progressing=True** after an upgrade. To fix this, set **AllowedSourceRanges** that overwrites the **spec.loadBalancerSourceRanges** parameter and clears the **service.beta.kubernetes.io/load-balancer-source-ranges** annotation. The Ingress Controller starts reporting **Progressing=False** again.

Prerequisites

- You have set the **service.beta.kubernetes.io/load-balancer-source-ranges** annotation.

Procedure

1. Check that the **service.beta.kubernetes.io/load-balancer-source-ranges** is set by entering the following command:

```
$ oc get svc router-default -n openshift-ingress -o yaml
```

Example output

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.beta.kubernetes.io/load-balancer-source-ranges: 192.168.0.1/32
```

2. Check that the **spec.loadBalancerSourceRanges** parameter is unset by entering the following command:

```
$ oc get svc router-default -n openshift-ingress -o yaml
```

Example output

```
...
spec:
  loadBalancerSourceRanges:
  - 0.0.0.0/0
...
```

3. Update your cluster to OpenShift Container Platform 4.18.

4. Set the allowed source ranges API for the **ingresscontroller** by running the following command:

```
$ oc -n openshift-ingress-operator patch ingresscontroller/default \
    --type=merge --patch='{"spec":{"endpointPublishingStrategy": \
    {"loadBalancer":{"allowedSourceRanges":["0.0.0.0/0"]}}}}'
```

where:

allowedSourceRanges

The example value **0.0.0.0/0** specifies the allowed source range.

2.9.3. Additional resources

- Introduction to OpenShift updates

2.10. PATCHING EXISTING INGRESS OBJECTS

You can update or modify the following fields of existing **Ingress** objects without recreating the objects or disrupting services to these objects:

- Specifications
- Host
- Path
- Backend services
- SSL/TLS settings
- Annotations

2.10.1. Patching Ingress objects to resolve an ingressWithoutClassName alert

To prevent certain routing issues, you must define define the **ingressClassName** field for each **Ingress** object.

NOTE

Approximately 24 hours after you create an **Ingress** object, the Ingress Controller sends you an **ingressWithoutClassName** alert to remind you to set the **ingressClassName** field.

The procedure demonstrates patching the **Ingress** objects with a completed **ingressClassName** field to ensure proper routing and functionality.

Procedure

1. List all **IngressClass** objects:

```
$ oc get ingressclass
```

2. List all **Ingress** objects in all namespaces:

```
$ oc get ingress -A
```

3. Patch the **Ingress** object by running the following command. This command patches the **Ingress** object to include the desired ingress class name.

```
$ oc patch ingress/<ingress_name> --type=merge --patch '{"spec":
{"ingressClassName":"openshift-default"}}'
```

- **<ingress_name>**: Replace **<ingress_name>** with the name of the **Ingress** object.

2.11. CONFIGURING AN INGRESS CONTROLLER FOR MANUAL DNS MANAGEMENT

To ensure application accessiblity across external networks in OpenShift Container Platform, you can manually configure DNS records for an Ingress Controller.

As a cluster administrator, when you create an Ingress Controller, the Operator manages the DNS records automatically. This has some limitations when the required DNS zone is different from the cluster DNS zone or when the DNS zone is hosted outside the cloud provider.

The following list details key aspects for a managed DNS management policy:

- The Managed DNS management policy for Ingress Controllers ensures that the lifecycle of the wildcard DNS record on the cloud provider is automatically managed by the Operator. This is the default behavior.
- When you change an Ingress Controller from **Managed** to **Unmanaged** DNS management policy, the Operator does not clean up the previous wildcard DNS record provisioned on the cloud.
- When you change an Ingress Controller from **Unmanaged** to **Managed** DNS management policy, the Operator attempts to create the DNS record on the cloud provider if it does not exist or updates the DNS record if it already exists.

The following list details key aspects for a unmanaged DNS management policy:

- The Unmanaged DNS management policy for Ingress Controllers ensures that the lifecycle of the wildcard DNS record on the cloud provider is not automatically managed; instead, it becomes the responsibility of the cluster administrator.

2.11.1. Creating a custom Ingress Controller with the Unmanaged DNS management policy

As a cluster administrator, you can create a new custom Ingress Controller with the Unmanaged DNS management policy.

Prerequisites

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

Procedure

1. Create a custom resource (CR) file named **sample-ingress.yaml** containing the following:

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  namespace: openshift-ingress-operator
  name: <name>
spec:
  domain: <domain>
  endpointPublishingStrategy:
    type: LoadBalancerService
    loadBalancer:
      scope: External
      dnsManagementPolicy: Unmanaged
```

where:

metadata.name

Specify the **<name>** with a name for the **IngressController** object.

spec.domain

Specify the **domain** based on the DNS record that was created as a prerequisite.

loadBalancer.scope

Specify the **scope** as **External** to expose the load balancer externally.
loadBalancer.dnsManagementPolicy: Specifies if the Ingress Controller is managing the lifecycle of the wildcard DNS record associated with the load balancer. The valid values are **Managed** and **Unmanaged**. The default value is **Managed**.

2. Save the file to apply the changes.

```
oc apply -f <name>.yaml 1
```

Inspect the output and confirm that **dnsManagementPolicy** is set to **Unmanaged**.

2.11.2. Modifying an existing Ingress Controller

As a cluster administrator, you can modify an existing Ingress Controller to manually manage the DNS record lifecycle.

Prerequisites

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

Procedure

1. Modify the chosen **IngressController** to set **dnsManagementPolicy**:

```
SCOPE=$(oc -n openshift-ingress-operator get ingresscontroller <name> -o=jsonpath="
{.status.endpointPublishingStrategy.loadBalancer.scope}")
```

```
oc -n openshift-ingress-operator patch ingresscontrollers/<name> --type=merge --
patch='{"spec":{"endpointPublishingStrategy":{"type":"LoadBalancerService","loadBalancer":
{"dnsManagementPolicy":"Unmanaged", "scope":"${SCOPE}"}}}}'
```

2. Optional: You can delete the associated DNS record in the cloud provider.

2.11.3. Additional resources

- Ingress Controller configuration parameters

CHAPTER 3. LOAD BALANCING ON RHOSP

To distribute network traffic and communications activity evenly across your compute instances in RHOSP, configure load balancing services.

3.1. LIMITATIONS OF LOAD BALANCER SERVICES

To optimise network resource management and mitigate operational risks in OpenShift Container Platform clusters on Red Hat OpenStack Platform (RHOSP), review the implementation of Octavia for load balancer services.

OpenShift Container Platform clusters on Red Hat OpenStack Platform (RHOSP) use Octavia to handle load balancer services. As a result, your cluster has several functional limitations.

RHOSP Octavia has two supported providers: Amphora and OVN. These providers differ in available features and implementation details. These distinctions affect load balancer services that you create on your cluster.

3.1.1. Local external traffic policies

You can set the external traffic policy (ETP) parameter, **.spec.externalTrafficPolicy**, on a load balancer service to preserve the source IP address of incoming traffic when it reaches service endpoint pods.

If your cluster uses the Amphora Octavia provider, the source IP of the traffic is replaced with the IP address of the Amphora VM. This behavior does not occur if your cluster uses the OVN Octavia provider.

Having the **ETP** option set to **Local** requires creating health monitors for the load balancer. Without health monitors, traffic can be routed to a node that does not have a functional endpoint, which causes the connection to drop. To force Cloud Provider OpenStack to create health monitors, you must set the value of the **create-monitor** option in the cloud provider configuration to **true**.

In RHOSP 16.2, the OVN Octavia provider does not support health monitors. Therefore, setting the ETP to **Local** is unsupported.

In RHOSP 16.2, the Amphora Octavia provider does not support HTTP monitors on UDP pools. As a result, UDP load balancer services have **UDP-CONNECT** monitors created instead. Due to implementation details, this configuration only functions properly with the OVN-Kubernetes CNI plugin.

3.2. SCALING CLUSTERS FOR APPLICATION TRAFFIC BY USING OCTAVIA

To distribute traffic across multiple virtual machines (VMs), configure your cluster that runs on Red Hat OpenStack Platform (RHOSP) to use the Octavia load balancing service. By using this feature, you can mitigate the bottleneck that single machines or addresses create.

You must create your own Octavia load balancer to use it for application network scaling.

3.2.1. Scaling clusters by using Octavia

To ensure high availability and distribute traffic across multiple cluster API access points in OpenShift Container Platform on RHOSP, create an Octavia load balancer. Configuring your cluster to use multiple balancers prevents network bottlenecks and ensures continuous access to your API services.

Prerequisites

- Octavia is available on your Red Hat OpenStack Platform (RHOSP) deployment.

Procedure

1. From the command-line interface (CLI), create an Octavia load balancer that uses the Amphora driver:

```
$ openstack loadbalancer create --name API_OCP_CLUSTER --vip-subnet-id
<id_of_worker_vms_subnet>
```

   You can use a name of your choice instead of **API_OCP_CLUSTER**.

2. After the load balancer becomes active, create listeners:

```
$ openstack loadbalancer listener create --name API_OCP_CLUSTER_6443 --protocol
HTTPS--protocol-port 6443 API_OCP_CLUSTER
```

NOTE

To view the status of the load balancer, enter **openstack loadbalancer list**.

3. Create a pool that uses the round-robin algorithm and has session persistence enabled:

```
$ openstack loadbalancer pool create --name API_OCP_CLUSTER_pool_6443 --lb-
algorithm ROUND_ROBIN --session-persistence type=<source_IP_address> --listener
API_OCP_CLUSTER_6443 --protocol HTTPS
```

4. To ensure that control-plane machines are available, create a health monitor:

```
$ openstack loadbalancer healthmonitor create --delay 5 --max-retries 4 --timeout 10 --type
TCP API_OCP_CLUSTER_pool_6443
```

5. Add the control plane machines as members of the load balancer pool:

```
$ for SERVER in $(MASTER-0-IP MASTER-1-IP MASTER-2-IP)
do
  openstack loadbalancer member create --address $SERVER  --protocol-port 6443
API_OCP_CLUSTER_pool_6443
done
```

6. Optional: To reuse the cluster API floating IP address, unset it:

```
$ openstack floating ip unset $API_FIP
```

7. Add either the unset **API_FIP** or a new address to the created load balancer VIP:

```
$ openstack floating ip set  --port $(openstack loadbalancer show -c <vip_port_id> -f value
API_OCP_CLUSTER) $API_FIP
```

   Your cluster now uses Octavia for load balancing.

3.3. SERVICES FOR A USER-MANAGED LOAD BALANCER

To integrate your infrastructure with existing network standards or gain more control over traffic management in OpenShift Container Platform on Red Hat OpenStack Platform (RHOSP) , configure services for a user-managed load balancer.

IMPORTANT

Configuring a user-managed load balancer depends on your vendor's load balancer.

The information and examples in this section are for guideline purposes only. Consult the vendor documentation for more specific information about the vendor's load balancer.

Red Hat supports the following services for a user-managed load balancer:

- Ingress Controller
- OpenShift API
- OpenShift MachineConfig API

You can choose whether you want to configure one or all of these services for a user-managed load balancer. Configuring only the Ingress Controller service is a common configuration option. To better understand each service, view the following diagrams:

Figure 3.1. Example network workflow that shows an Ingress Controller operating in an OpenShift Container Platform environment

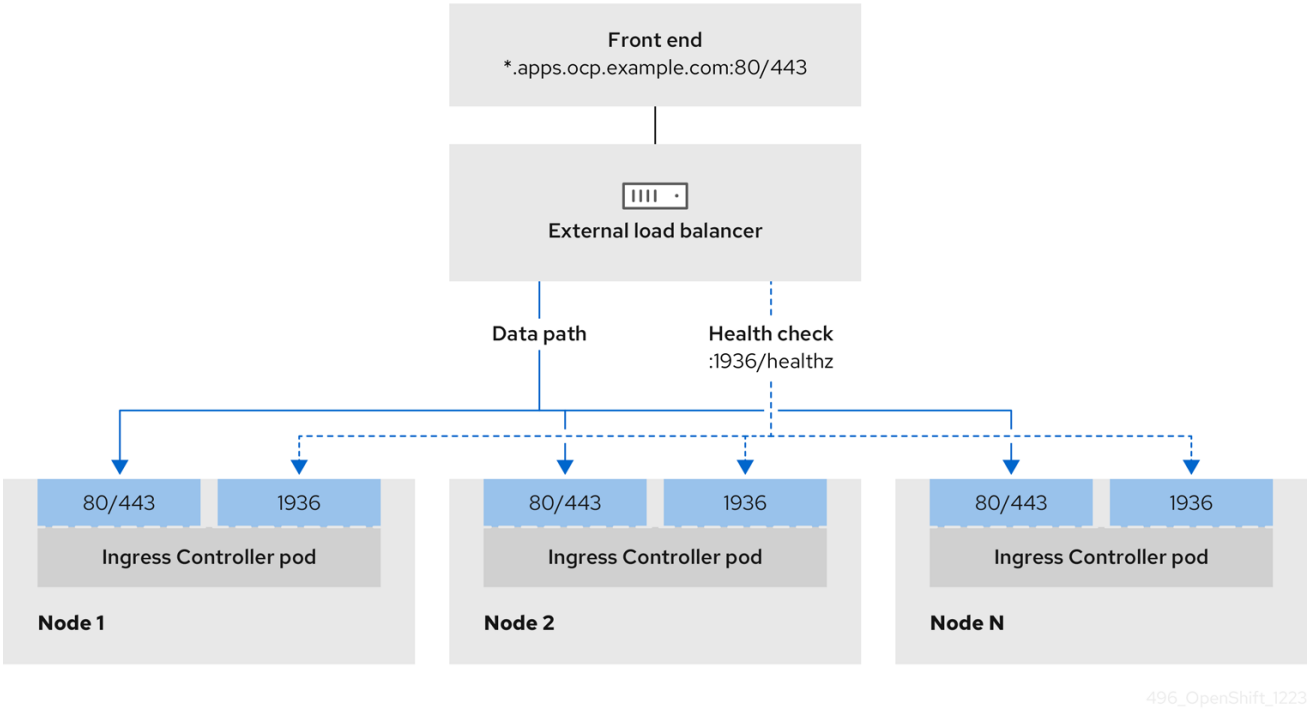


Figure 3.2. Example network workflow that shows an OpenShift API operating in an OpenShift Container Platform environment

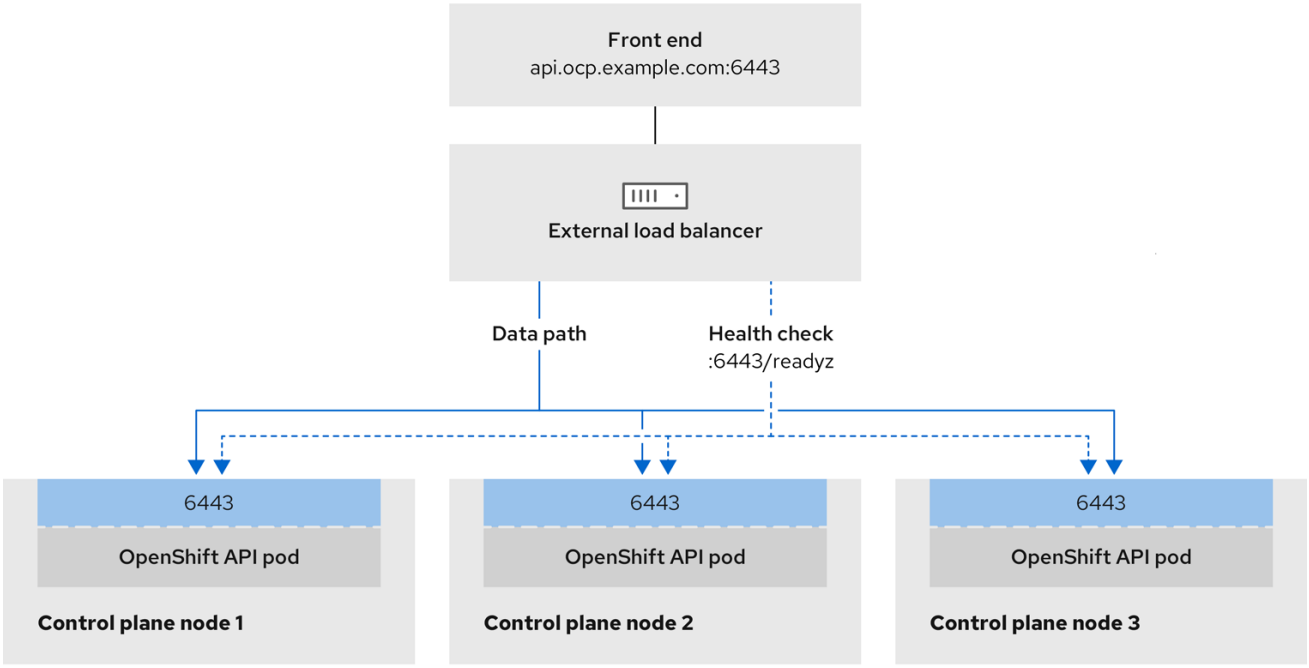
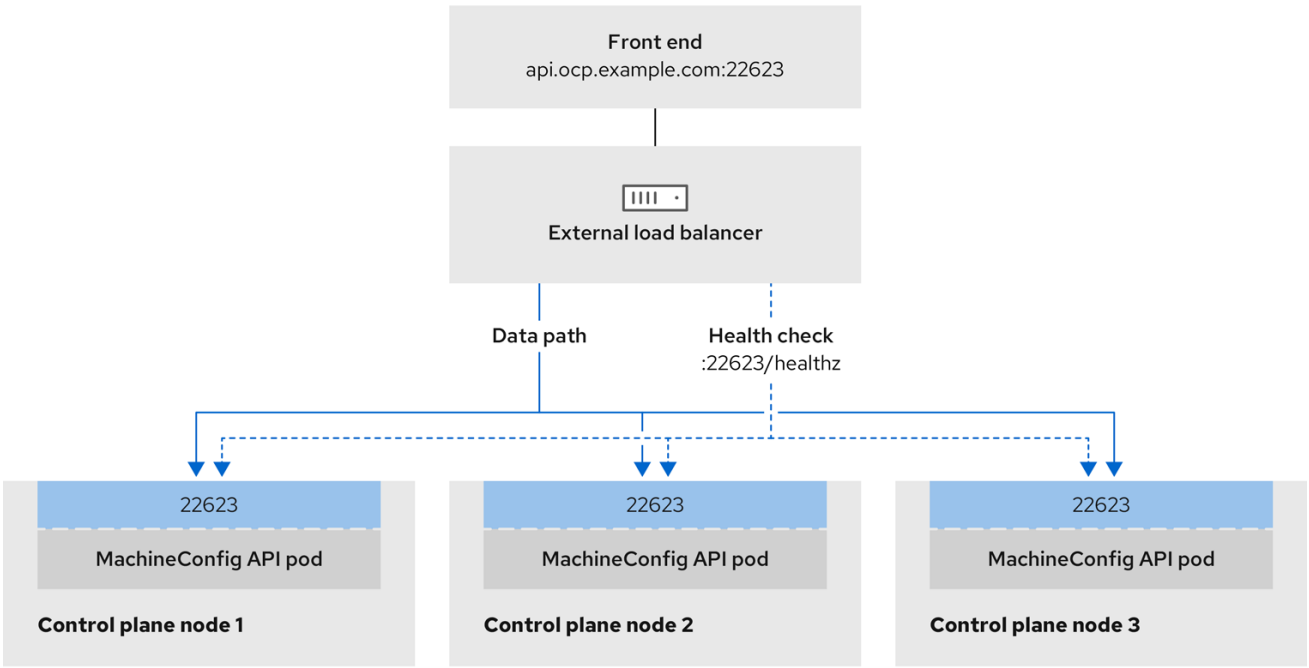


Figure 3.3. Example network workflow that shows an OpenShift MachineConfig API operating in an OpenShift Container Platform environment



The following configuration options are supported for user-managed load balancers:

- Use a node selector to map the Ingress Controller to a specific set of nodes. You must assign a static IP address to each node in this set, or configure each node to receive the same IP address from the Dynamic Host Configuration Protocol (DHCP). Infrastructure nodes commonly receive this type of configuration.

- Target all IP addresses on a subnet. This configuration can reduce maintenance overhead, because you can create and destroy nodes within those networks without reconfiguring the load balancer targets. If you deploy your ingress pods by using a machine set on a smaller network, such as a /**27** or /**28**, you can simplify your load balancer targets.

  TIP

  You can list all IP addresses that exist in a network by checking the machine config pool's resources.

Before you configure a user-managed load balancer for your OpenShift Container Platform cluster, consider the following information:

- For a front-end IP address, you can use the same IP address for the front-end IP address, the Ingress Controller's load balancer, and API load balancer. Check the vendor's documentation for this capability.
- For a back-end IP address, ensure that an IP address for an OpenShift Container Platform control plane node does not change during the lifetime of the user-managed load balancer. You can achieve this by completing one of the following actions:
  - Assign a static IP address to each control plane node.
  - Configure each node to receive the same IP address from the DHCP every time the node requests a DHCP lease. Depending on the vendor, the DHCP lease might be in the form of an IP reservation or a static DHCP assignment.
- Manually define each node that runs the Ingress Controller in the user-managed load balancer for the Ingress Controller back-end service. For example, if the Ingress Controller moves to an undefined node, a connection outage can occur.

3.3.1. Configuring a user-managed load balancer

To integrate your infrastructure with existing network standards or gain more control over traffic management in OpenShift Container Platform on Red Hat OpenStack Platform (RHOSP) , use a user-managed load balancer in place of the default load balancer.

IMPORTANT

Before you configure a user-managed load balancer, ensure that you read the "Services for a user-managed load balancer" section.

Read the following prerequisites that apply to the service that you want to configure for your user-managed load balancer.

NOTE

MetalLB, which runs on a cluster, functions as a user-managed load balancer.

Prerequisites

The following list details OpenShift API prerequisites:

- You defined a front-end IP address.

- TCP ports 6443 and 22623 are exposed on the front-end IP address of your load balancer. Check the following items:
  - Port 6443 provides access to the OpenShift API service.
  - Port 22623 can provide ignition startup configurations to nodes.
- The front-end IP address and port 6443 are reachable by all users of your system with a location external to your OpenShift Container Platform cluster.
- The front-end IP address and port 22623 are reachable only by OpenShift Container Platform nodes.
- The load balancer backend can communicate with OpenShift Container Platform control plane nodes on port 6443 and 22623.

The following list details Ingress Controller prerequisites:

- You defined a front-end IP address.
- TCP port 443 and port 80 are exposed on the front-end IP address of your load balancer.
- The front-end IP address, port 80 and port 443 are reachable by all users of your system with a location external to your OpenShift Container Platform cluster.
- The front-end IP address, port 80 and port 443 are reachable by all nodes that operate in your OpenShift Container Platform cluster.
- The load balancer backend can communicate with OpenShift Container Platform nodes that run the Ingress Controller on ports 80, 443, and 1936.

The following list details prerequisites for health check URL specifications:

You can configure most load balancers by setting health check URLs that determine if a service is available or unavailable. OpenShift Container Platform provides these health checks for the OpenShift API, Machine Configuration API, and Ingress Controller backend services.

The following example shows a Kubernetes API health check specification for a backend service:

```
Path: HTTPS:6443/readyz
Healthy threshold: 2
Unhealthy threshold: 2
Timeout: 10
Interval: 10
```

The following example shows a Machine Config API health check specification for a backend service:

```
Path: HTTPS:22623/healthz
Healthy threshold: 2
Unhealthy threshold: 2
Timeout: 10
Interval: 10
```

The following example shows a Ingress Controller health check specification for a backend service:

```
Path: HTTP:1936/healthz/ready
```

Healthy threshold: 2
Unhealthy threshold: 2
Timeout: 5
Interval: 10

Procedure

1. Configure the HAProxy Ingress Controller, so that you can enable access to the cluster from your load balancer on ports 6443, 22623, 443, and 80. Depending on your needs, you can specify the IP address of a single subnet or IP addresses from multiple subnets in your HAProxy configuration.

   Example HAProxy configuration with one listed subnet

```
# ...
listen my-cluster-api-6443
    bind 192.168.1.100:6443
    mode tcp
    balance roundrobin
  option httpchk
  http-check connect
  http-check send meth GET uri /readyz
  http-check expect status 200
    server my-cluster-master-2 192.168.1.101:6443 check inter 10s rise 2 fall 2
    server my-cluster-master-0 192.168.1.102:6443 check inter 10s rise 2 fall 2
    server my-cluster-master-1 192.168.1.103:6443 check inter 10s rise 2 fall 2

listen my-cluster-machine-config-api-22623
    bind 192.168.1.100:22623
    mode tcp
    balance roundrobin
  option httpchk
  http-check connect
  http-check send meth GET uri /healthz
  http-check expect status 200
    server my-cluster-master-2 192.168.1.101:22623 check inter 10s rise 2 fall 2
    server my-cluster-master-0 192.168.1.102:22623 check inter 10s rise 2 fall 2
    server my-cluster-master-1 192.168.1.103:22623 check inter 10s rise 2 fall 2

listen my-cluster-apps-443
    bind 192.168.1.100:443
    mode tcp
    balance roundrobin
  option httpchk
  http-check connect
  http-check send meth GET uri /healthz/ready
  http-check expect status 200
    server my-cluster-worker-0 192.168.1.111:443 check port 1936 inter 10s rise 2 fall 2
    server my-cluster-worker-1 192.168.1.112:443 check port 1936 inter 10s rise 2 fall 2
    server my-cluster-worker-2 192.168.1.113:443 check port 1936 inter 10s rise 2 fall 2

listen my-cluster-apps-80
   bind 192.168.1.100:80
   mode tcp
   balance roundrobin
  option httpchk
```

```
  http-check connect
  http-check send meth GET uri /healthz/ready
  http-check expect status 200
    server my-cluster-worker-0 192.168.1.111:80 check port 1936 inter 10s rise 2 fall 2
    server my-cluster-worker-1 192.168.1.112:80 check port 1936 inter 10s rise 2 fall 2
    server my-cluster-worker-2 192.168.1.113:80 check port 1936 inter 10s rise 2 fall 2
# ...
```

Example HAProxy configuration with multiple listed subnets

```
# ...
listen api-server-6443
    bind *:6443
    mode tcp
      server master-00 192.168.83.89:6443 check inter 1s
      server master-01 192.168.84.90:6443 check inter 1s
      server master-02 192.168.85.99:6443 check inter 1s
      server bootstrap 192.168.80.89:6443 check inter 1s

listen machine-config-server-22623
    bind *:22623
    mode tcp
      server master-00 192.168.83.89:22623 check inter 1s
      server master-01 192.168.84.90:22623 check inter 1s
      server master-02 192.168.85.99:22623 check inter 1s
      server bootstrap 192.168.80.89:22623 check inter 1s

listen ingress-router-80
    bind *:80
    mode tcp
    balance source
      server worker-00 192.168.83.100:80 check inter 1s
      server worker-01 192.168.83.101:80 check inter 1s

listen ingress-router-443
    bind *:443
    mode tcp
    balance source
      server worker-00 192.168.83.100:443 check inter 1s
      server worker-01 192.168.83.101:443 check inter 1s

listen ironic-api-6385
    bind *:6385
    mode tcp
    balance source
      server master-00 192.168.83.89:6385 check inter 1s
      server master-01 192.168.84.90:6385 check inter 1s
      server master-02 192.168.85.99:6385 check inter 1s
      server bootstrap 192.168.80.89:6385 check inter 1s

listen inspector-api-5050
    bind *:5050
    mode tcp
    balance source
      server master-00 192.168.83.89:5050 check inter 1s
      server master-01 192.168.84.90:5050 check inter 1s
```

```
    server master-02 192.168.85.99:5050 check inter 1s
    server bootstrap 192.168.80.89:5050 check inter 1s
# ...
```

2. Use the **curl** CLI command to verify that the user-managed load balancer and its resources are operational:

   a. Verify that the cluster machine configuration API is accessible to the Kubernetes API server resource, by running the following command and observing the response:

   ```
   $ curl https://<loadbalancer_ip_address>:6443/version --insecure
   ```

   If the configuration is correct, you receive a JSON object in response:

   ```
   {
     "major": "1",
     "minor": "11+",
     "gitVersion": "v1.11.0+ad103ed",
     "gitCommit": "ad103ed",
     "gitTreeState": "clean",
     "buildDate": "2019-01-09T06:44:10Z",
     "goVersion": "go1.10.3",
     "compiler": "gc",
     "platform": "linux/amd64"
   }
   ```

   b. Verify that the cluster machine configuration API is accessible to the Machine config server resource, by running the following command and observing the output:

   ```
   $ curl -v https://<loadbalancer_ip_address>:22623/healthz --insecure
   ```

   If the configuration is correct, the output from the command shows the following response:

   ```
   HTTP/1.1 200 OK
   Content-Length: 0
   ```

   c. Verify that the controller is accessible to the Ingress Controller resource on port 80, by running the following command and observing the output:

   ```
   $ curl -I -L -H "Host: console-openshift-console.apps.<cluster_name>.<base_domain>"
   http://<load_balancer_front_end_IP_address>
   ```

   If the configuration is correct, the output from the command shows the following response:

   ```
   HTTP/1.1 302 Found
   content-length: 0
   location: https://console-openshift-console.apps.ocp4.private.opequon.net/
   cache-control: no-cache
   ```

   d. Verify that the controller is accessible to the Ingress Controller resource on port 443, by running the following command and observing the output:

```
$ curl -I -L --insecure --resolve console-openshift-console.apps.<cluster_name>.
<base_domain>:443:<Load Balancer Front End IP Address> https://console-openshift-
console.apps.<cluster_name>.<base_domain>
```

If the configuration is correct, the output from the command shows the following response:

```
HTTP/1.1 200 OK
referrer-policy: strict-origin-when-cross-origin
set-cookie: csrf-
token=UlYWOyQ62LWjw2h003xtYSKlh1a0Py2hhctw0WmV2YEdhJjFyQwWcGBsja261dG
LgaYO0nxzVErhiXt6QepA7g==; Path=/; Secure; SameSite=Lax
x-content-type-options: nosniff
x-dns-prefetch-control: off
x-frame-options: DENY
x-xss-protection: 1; mode=block
date: Wed, 04 Oct 2023 16:29:38 GMT
content-type: text/html; charset=utf-8
set-cookie:
1e2670d92730b515ce3a1bb65da45062=1bf5e9573c9a2760c964ed1659cc1673; path=/;
HttpOnly; Secure; SameSite=None
cache-control: private
```

3. Configure the DNS records for your cluster to target the front-end IP addresses of the user-managed load balancer. You must update records to your DNS server for the cluster API and applications over the load balancer. The following examples shows modified DNS records:

```
<load_balancer_ip_address>  A  api.<cluster_name>.<base_domain>
A record pointing to Load Balancer Front End
```

```
<load_balancer_ip_address>   A apps.<cluster_name>.<base_domain>
A record pointing to Load Balancer Front End
```

IMPORTANT

DNS propagation might take some time for each DNS record to become available. Ensure that each DNS record propagates before validating each record.

4. For your OpenShift Container Platform cluster to use the user-managed load balancer, you must specify the following configuration in your cluster's **install-config.yaml** file:

```
# ...
platform:
  openstack:
    loadBalancer:
      type: UserManaged
    apiVIPs:
    - <api_ip> 1
    ingressVIPs:
    - <ingress_ip> 2
# ...
```

where:

loadBalancer.type

Set **UserManaged** for the **type** parameter to specify a user-managed load balancer for your cluster. The parameter defaults to **OpenShiftManagedDefault**, which denotes the default internal load balancer. For services defined in an **openshift-kni-infra** namespace, a user-managed load balancer can deploy the **coredns** service to pods in your cluster but ignores **keepalived** and **haproxy** services.

loadBalancer.<api_ip>

Specifies a user-managed load balancer. Specify the user-managed load balancer's public IP address, so that the Kubernetes API can communicate with the user-managed load balancer. Mandatory parameter.

loadBalancer.<ingress_ip>

Specifies a user-managed load balancer. Specify the user-managed load balancer's public IP address, so that the user-managed load balancer can manage ingress traffic for your cluster. Mandatory parameter.

Verification

1. Use the **curl** CLI command to verify that the user-managed load balancer and DNS record configuration are operational:

   a. Verify that you can access the cluster API, by running the following command and observing the output:

   ```
   $ curl https://api.<cluster_name>.<base_domain>:6443/version --insecure
   ```

   If the configuration is correct, you receive a JSON object in response:

   ```
   {
     "major": "1",
     "minor": "11+",
     "gitVersion": "v1.11.0+ad103ed",
     "gitCommit": "ad103ed",
     "gitTreeState": "clean",
     "buildDate": "2019-01-09T06:44:10Z",
     "goVersion": "go1.10.3",
     "compiler": "gc",
     "platform": "linux/amd64"
     }
   ```

   b. Verify that you can access the cluster machine configuration, by running the following command and observing the output:

   ```
   $ curl -v https://api.<cluster_name>.<base_domain>:22623/healthz --insecure
   ```

   If the configuration is correct, the output from the command shows the following response:

   ```
   HTTP/1.1 200 OK
   Content-Length: 0
   ```

   c. Verify that you can access each cluster application on port 80, by running the following command and observing the output:

```
$ curl http://console-openshift-console.apps.<cluster_name>.<base_domain> -I -L --
insecure
```

If the configuration is correct, the output from the command shows the following response:

```
HTTP/1.1 302 Found
content-length: 0
location: https://console-openshift-console.apps.<cluster-name>.<base domain>/
cache-control: no-cacheHTTP/1.1 200 OK
referrer-policy: strict-origin-when-cross-origin
set-cookie: csrf-
token=39HoZgztDnzjJkq/JuLJMeoKNXlfiVv2YgZc09c3TBOBU4NI6kDXaJH1LdicNhN1UsQ
Wzon4Dor9GWGfopaTEQ==; Path=/; Secure
x-content-type-options: nosniff
x-dns-prefetch-control: off
x-frame-options: DENY
x-xss-protection: 1; mode=block
date: Tue, 17 Nov 2020 08:42:10 GMT
content-type: text/html; charset=utf-8
set-cookie:
1e2670d92730b515ce3a1bb65da45062=9b714eb87e93cf34853e87a92d6894be; path=/;
HttpOnly; Secure; SameSite=None
cache-control: private
```

d. Verify that you can access each cluster application on port 443, by running the following command and observing the output:

```
$ curl https://console-openshift-console.apps.<cluster_name>.<base_domain> -I -L --
insecure
```

If the configuration is correct, the output from the command shows the following response:

```
HTTP/1.1 200 OK
referrer-policy: strict-origin-when-cross-origin
set-cookie: csrf-
token=UlYWOyQ62LWjw2h003xtYSKlh1a0Py2hhctw0WmV2YEdhJjFyQwWcGBsja261dG
LgaYO0nxzVErhiXt6QepA7g==; Path=/; Secure; SameSite=Lax
x-content-type-options: nosniff
x-dns-prefetch-control: off
x-frame-options: DENY
x-xss-protection: 1; mode=block
date: Wed, 04 Oct 2023 16:29:38 GMT
content-type: text/html; charset=utf-8
set-cookie:
1e2670d92730b515ce3a1bb65da45062=1bf5e9573c9a2760c964ed1659cc1673; path=/;
HttpOnly; Secure; SameSite=None
cache-control: private
```

3.4. SPECIFYING A FLOATING IP ADDRESS IN THE INGRESS CONTROLLER

To establish external access to your OpenShift Container Platform cluster on Red Hat OpenStack Platform (RHOSP), use the automatically assigned floating IP address. The floating IP address is associated with your Ingress port.

You might want to precreate a floating IP address before updating your DNS records and cluster deployment. In this situation, you can define a floating IP address to the Ingress Controller. You can do this regardless of whether you are using Octavia or a user-managed cluster.

Procedure

1. Create the Ingress Controller custom resource (CR) file with the floating IPs:

   Example Ingress config sample-ingress.yaml

   ```
   apiVersion: operator.openshift.io/v1
   kind: IngressController
   metadata:
     namespace: openshift-ingress-operator
     name: <name>
   spec:
     domain: <domain>
     endpointPublishingStrategy:
       type: LoadBalancerService
       loadBalancer:
         scope: External
         providerParameters:
           type: OpenStack
           openstack:
             floatingIP: <ingress_port_IP>
   ```

   where:

   metadata.name

   Specifies the name of your Ingress Controller. If you are using the default Ingress Controller, the value for this field is **default**.

   spec.domain

   Specifies the DNS name serviced by the Ingress Controller.

   loadBalancer.scope

   You must set the scope to **External** to use a floating IP address.

   openstack.floatingIP

   Specifies the floating IP address associated with the port your Ingress Controller is listening on.

2. Apply the CR file by running the following command:

   ```
   $ oc apply -f sample-ingress.yaml
   ```

3. Update your DNS records with the Ingress Controller endpoint:

   ```
   *.apps.<name>.<domain>. IN A <ingress_port_IP>
   ```

4. Continue with creating your OpenShift Container Platform cluster.

Verification

- Confirm that the load balancer was successfully provisioned by checking the **IngressController** conditions using the following command:

```
$ oc get ingresscontroller -n openshift-ingress-operator <name> -o jsonpath="
{.status.conditions}" | yq -PC
```

CHAPTER 4. LOAD BALANCING WITH METALLB

4.1. CONFIGURING METALLB ADDRESS POOLS

To allocate and manage the IP addresses assigned to load balancer services, configure MetalLB address pool custom resources. Defining these pools ensures that application workloads remain reachable through designated network ranges for consistent external access.

The namespaces used in the examples show **metallb-system** as the namespace.

For more information about how to install the MetalLB Operator, see About MetalLB and the MetalLB Operator.

4.1.1. About the IPAddressPool custom resource

To define the IP address ranges available for load balancer services, configure the properties of the MetalLB **IPAddressPool** custom resource (CR). Establishing these parameters allows your cluster to automatically allocate specific external addresses to your application workloads.

The following table details the parameters for the **IPAddressPool** CR:

Table 4.1. MetalLB IPAddressPool pool custom resource

Parameter	Type	Description
metadata.name	**string**	Specifies the name for the address pool. When you add a service, you can specify this pool name in the **metallb.io/address-pool** annotation to select an IP address from a specific pool. The names **doc-example**, **silver**, and **gold** are used throughout the documentation.
metadata.namespace	**string**	Specifies the namespace for the address pool. Specify the same namespace that the MetalLB Operator uses.
metadata.label	**string**	Optional: Specifies the key-value pair assigned to the **IPAddressPool**. This can be referenced by the **ipAddressPoolSelectors** in the **BGPAdvertisement** and **L2Advertisement** CRD to associate the **IPAddressPool** with the advertisement
spec.addresses	**string**	Specifies a list of IP addresses for the MetalLB Operator to assign to services. You can specify multiple ranges in a single pool, where these ranges all share the same settings. Specify each range in Classless Inter-Domain Routing (CIDR) notation or as starting and ending IP addresses separated with a hyphen.

Parameter	Type	Description
spec.autoAssign	**boolean**	Optional: Specifies whether the MetalLB Operator automatically assigns IP addresses from this pool. Specify **false** if you want to explicitly request an IP address from this pool with the **metallb.io/address-pool** annotation. The default value is **true**. NOTE For IP address pool configurations, ensure the addresses parameter specifies only IP addresses that are available and not in use by other network devices, especially gateway addresses, to prevent conflicts when **autoAssign** is enabled.
spec.avoidBuggyIPs	**boolean**	Optional: When you set the parameter to enabled, the IP addresses ending **.0** and **.255** are not allocated from the pool. The default value is **false**. Some older consumer network equipment mistakenly block IP addresses ending in **.0** and **.255**.

You can assign IP addresses from an **IPAddressPool** to services and namespaces by configuring the **spec.serviceAllocation** specification.

Table 4.2. MetalLB IPAddressPool custom resource spec.serviceAllocation subfields

Parameter	Type	Description
priority	**int**	Optional: Defines the priority between IP address pools when more than one IP address pool matches a service or namespace. A lower number indicates a higher priority.
namespaces	**array (string)**	Optional: Specifies a list of namespaces that you can assign to IP addresses in an IP address pool.
namespaceSelectors	**array (LabelSelector)**	Optional: Specifies namespace labels that you can assign to IP addresses from an IP address pool by using label selectors in a list format.
serviceSelectors	**array (LabelSelector)**	Optional: Specifies service labels that you can assign to IP addresses from an address pool by using label selectors in a list format.

4.1.2. Configuring an address pool

To precisely manage external access to application workloads, configure MetalLB address pools for your cluster. By defining these pools, you can control the specific IP address ranges assigned to load balancer services for consistent network routing.

Prerequisites

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

Procedure

1. Create a file, such as **ipaddresspool.yaml**, with content like the following example:

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  namespace: metallb-system
  name: doc-example
  labels:
    zone: east
spec:
  addresses:
  - 203.0.113.1-203.0.113.10
  - 203.0.113.65-203.0.113.75
# ...
```

where:

labels

The label assigned to the **IPAddressPool** can be referenced by the **ipAddressPoolSelectors** in the **BGPAdvertisement** CRD to associate the **IPAddressPool** with the advertisement.

2. Apply the configuration for the IP address pool by entering the following command:

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

Verification

1. View the address pool by entering the following command:

```
$ oc describe -n metallb-system IPAddressPool doc-example
```

Example output

```
Name:         doc-example
Namespace:    metallb-system
Labels:       zone=east
Annotations:  <none>
API Version:  metallb.io/v1beta1
Kind:         IPAddressPool
Metadata:
  ...
Spec:
  Addresses:
    203.0.113.1-203.0.113.10
```

```
    203.0.113.65-203.0.113.75
  Auto Assign:  true
Events:         <none>
```

2. Confirm that the address pool name, such as **doc-example**, and the IP address ranges exist in the output.

4.1.3. Configure MetalLB address pool for VLAN

To precisely manage external access across a specific VLAN, configure MetalLB address pools for your cluster. Defining these pools ensures that load balancer services receive authorized IP addresses from designated network ranges for secure and consistent routing.

Prerequisites

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

Procedure

1. Create a file, such as **ipaddresspool-vlan.yaml**, that is similar to the following example:

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  namespace: metallb-system
  name: doc-example-vlan
  labels:
    zone: east
spec:
  addresses:
  - 192.168.100.1-192.168.100.254
# ...
```

where:

labels.zone

This label assigned to the **IPAddressPool** can be referenced by the **ipAddressPoolSelectors** in the **BGPAdvertisement** CRD to associate the **IPAddressPool** with the advertisement.

spec.addresses

This IP range must match the subnet assigned to the VLAN on your network. To support layer 2 (L2) mode, the IP address range must be within the same subnet as the cluster nodes.

2. Apply the configuration for the IP address pool:

```
$ oc apply -f ipaddresspool-vlan.yaml
```

3. To ensure this configuration applies to the VLAN, you need to set the **spec gatewayConfig.ipForwarding** to **Global**.

   a. Run the following command to edit the network configuration custom resource (CR):

   ```
   $ oc edit network.operator.openshift/cluster
   ```

   b. Update the **spec.defaultNetwork.ovnKubernetesConfig** section to include the **gatewayConfig.ipForwarding** set to **Global**. The following example demonstrates this configuration:

   ```
   apiVersion: operator.openshift.io/v1
   kind: Network
   metadata:
     name: cluster
   spec:
     clusterNetwork:
       - cidr: 10.128.0.0/14
         hostPrefix: 23
     defaultNetwork:
       type: OVNKubernetes
       ovnKubernetesConfig:
         gatewayConfig:
           ipForwarding: Global
   # ...
   ```

4.1.4. Example address pool configurations

To precisely allocate IP address ranges for cluster services, configure MetalLB address pools by using Classless Inter-Domain Routing (CIDR) notation or hyphenated bounds. Defining these specific ranges ensures that application workloads receive valid IP assignments that align with your existing network infrastructure requirements.

Example of IPv4 and CIDR ranges

You can specify a range of IP addresses in classless inter-domain routing (CIDR) notation. You can combine CIDR notation with the notation that uses a hyphen to separate lower and upper bounds.

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: doc-example-cidr
  namespace: metallb-system
spec:
  addresses:
  - 192.168.100.0/24
  - 192.168.200.0/24
  - 192.168.255.1-192.168.255.5
# ...
```

Example of assigning IP addresses

You can set the **autoAssign** parameter to **false** to prevent MetalLB from automatically assigning IP addresses from the address pool. You can then assign a single IP address or multiple IP addresses from an IP address pool. To assign an IP address, append the **/32** CIDR notation to the target IP

address in the **spec.addresses** parameter. This setting ensures that only the specific IP address is available for assignment, leaving non-reserved IP addresses for application use.

Example IPAddressPool CR that assigns multiple IP addresses

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: doc-example-reserved
  namespace: metallb-system
spec:
  addresses:
  - 192.168.100.1/32
  - 192.168.200.1/32
  autoAssign: false
# ...
```

NOTE

When you add a service, you can request a specific IP address from the address pool or you can specify the pool name in an annotation to request any IP address from the pool.

Example of IPv4 and IPv6 addresses

You can add address pools that use IPv4 and IPv6. You can specify multiple ranges in the **addresses** list, just like several IPv4 examples.

How the service is assigned to a single IPv4 address, a single IPv6 address, or both is determined by how you add the service. The **spec.ipFamilies** and **spec.ipFamilyPolicy** parameters control how IP addresses are assigned to the service.

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: doc-example-combined
  namespace: metallb-system
spec:
  addresses:
  - 10.0.100.0/28 (1)
  - 2002:2:2::1-2002:2:2::100
# ...
```

spec.addresses: Where **10.0.100.0/28** is the local network IP address followed by the **/28** network prefix.

Example of assigning IP address pools to services or namespaces

You can assign IP addresses from an **IPAddressPool** to services and namespaces that you specify.

If you assign a service or namespace to more than one IP address pool, MetalLB uses an available IP address from the higher-priority IP address pool. If no IP addresses are available from the assigned IP address pools with a high priority, MetalLB uses available IP addresses from an IP address pool with lower priority or no priority.

NOTE

You can use the **matchLabels** label selector, the **matchExpressions** label selector, or both, for the **namespaceSelectors** and **serviceSelectors** specifications. This example demonstrates one label selector for each specification.

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: doc-example-service-allocation
  namespace: metallb-system
spec:
  addresses:
    - 192.168.20.0/24
  serviceAllocation:
    priority: 50
    namespaces:
      - namespace-a
      - namespace-b
    namespaceSelectors:
      - matchLabels:
          zone: east
    serviceSelectors:
      - matchExpressions:
        - key: security
          operator: In
          values:
          - S1
# ...
```

where:

serviceAllocation.priority

Assign a priority to the address pool. A lower number indicates a higher priority.

serviceAllocation.namespaces

Assign one or more namespaces to the IP address pool in a list format.

serviceAllocation.namespaceSelectors

Assign one or more namespace labels to the IP address pool by using label selectors in a list format.

serviceAllocation.serviceSelectors

Assign one or more service labels to the IP address pool by using label selectors in a list format.

4.1.5. Additional resources

- Configuring MetalLB with an L2 advertisement and label
- Configuring MetalLB BGP peers
- Configuring services to use MetalLB

4.2. ABOUT ADVERTISING FOR THE IP ADDRESS POOLS

To provide fault-tolerant external IP addresses and load balancing for cluster services, configure

MetalLB to advertise addresses by using Layer 2 protocols, the Border Gateway Protocol (BGP), or both. Selecting the appropriate protocol ensures reliable traffic routing and high availability for your application workloads.

MetalLB supports advertising by using Layer 2 and BGP for the same set of IP addresses.

MetalLB provides the flexibility to assign address pools to specific BGP peers, effectively limiting advertising to a subset of nodes on the network. This allows for more complex configurations, such as facilitating the isolation of nodes or the segmentation of the network.

4.2.1. About the BGPAdvertisement custom resource

To configure how the cluster announces IP addresses to external peers, define the properties of the **BGPAdvertisement** custom resource (CR). Specifying these parameters ensures that MetalLB correctly manages routing advertisements for your application services within the network.

The following table describes the parameters for the **BGPAdvertisements** CR:

Table 4.3. BGPAdvertisements configuration

Parameter	Type	Description
metadata.name	**string**	Specifies the name for the BGP advertisement.
metadata.namespace	**string**	Specifies the namespace for the BGP advertisement. Specify the same namespace that the MetalLB Operator uses.
spec.aggregationLength	**integer**	Optional: Specifies the number of bits to include in a 32-bit CIDR mask. To aggregate the routes that the speaker advertises to BGP peers, the mask is applied to the routes for several service IP addresses and the speaker advertises the aggregated route. For example, with an aggregation length of **24**, the speaker can aggregate several **10.0.1.x/32** service IP addresses and advertise a single **10.0.1.0/24** route.
spec.aggregationLengthV6	**integer**	Optional: Specifies the number of bits to include in a 128-bit CIDR mask. For example, with an aggregation length of **124**, the speaker can aggregate several **fc00:f853:0ccd:e799::x/128** service IP addresses and advertise a single **fc00:f853:0ccd:e799::0/124** route.

Parameter	Type	Description
spec.communities	**string**	Optional: Specifies one or more BGP communities. Each community is specified as two 16-bit values separated by the colon character. Well-known communities must be specified as 16-bit values: • **NO_EXPORT**: **65535:65281** • **NO_ADVERTISE**: **65535:65282** • **NO_EXPORT_SUBCONFED**: **65535:65283**  NOTE You can also use community objects that are created along with the strings.
spec.localPref	**integer**	Optional: Specifies the local preference for this advertisement. This BGP attribute applies to BGP sessions within the Autonomous System.
spec.ipAddressPools	**string**	Optional: The list of **IPAddressPools** to advertise with this advertisement, selected by name.
spec.ipAddressPoolSelectors	**string**	Optional: A selector for the **IPAddressPools** that gets advertised with this advertisement. This is for associating the **IPAddressPool** to the advertisement based on the label assigned to the **IPAddressPool** instead of the name itself. If no **IPAddressPool** is selected by this or by the list, the advertisement is applied to all the **IPAddressPools**.
spec.nodeSelectors	**string**	Optional: By setting the **NodeSelectors** parameter, you can limit the nodes to announce as next hops for the load balancer IP. When empty, all the nodes are announced as next hops.
spec.peers	**string**	Optional: Use a list to specify the **metadata.name** values for each **BGPPeer** resource that receives advertisements for the MetalLB service IP address. The MetalLB service IP address is assigned from the IP address pool. By default, the MetalLB service IP address is advertised to all configured **BGPPeer** resources. Set this parameter to limit the advertisement to specific **BGPpeer** resources.

4.2.2. Configure MetalLB with a BGP advertisement and a basic use case

To advertise specific IPv4 and IPv6 routes to peer BGP routers for assigned load-balancer IP addresses, configure MetalLB by using default preference and community settings. Establishing these routes ensures reliable external traffic delivery and consistent network propagation for your application

services.

Ensure that you can configure MetalLB so that the peer BGP routers receive one **203.0.113.200/32** route and one **fc00:f853:ccd:e799::1/128** route for each load-balancer IP address that MetalLB assigns to a service. If you do not specify the **localPref** and **communities** parameters, MetalLB advertises the routes with **localPref** set to `0 and no BGP communities.

4.2.2.1. Advertising a basic address pool configuration with BGP

To ensure application services are reachable from external network peers, configure MetalLB to advertise an **IPAddressPool** by using the BGP advertisement. Establishing this advertisement allows the external network to correctly route traffic to the load balancer IP addresses of your cluster services.

The procedure demonstrates how to configure MetalLB to advertise the **IPAddressPool** by using BGP.

Prerequisites

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

Procedure

1. Create an IP address pool.
   a. Create a file, such as **ipaddresspool.yaml**, with content like the following example:

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  namespace: metallb-system
  name: doc-example-bgp-basic
spec:
  addresses:
    - 203.0.113.200/30
    - fc00:f853:ccd:e799::/124
# ...
```

   b. Apply the configuration for the IP address pool:

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

2. Create a BGP advertisement.
   a. Create a file, such as **bgpadvertisement.yaml**, with content like the following example:

```
apiVersion: metallb.io/v1beta1
kind: BGPAdvertisement
metadata:
  name: bgpadvertisement-basic
  namespace: metallb-system
spec:
```

```
  ipAddressPools:
  - doc-example-bgp-basic
# ...
```

b. Apply the configuration:

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

4.2.3. Configuring MetalLB with a BGP advertisement and an advanced use case

To assign application services specific IP addresses from designated IPv4 and IPv6 ranges, configure MetalLB for advanced address allocation. Establishing these ranges and BGP advertisements ensures that your load-balancer services remain reachable through predictable network paths.

Configure MetalLB so that MetalLB assigns IP addresses to load-balancer services in the ranges between **203.0.113.200** and **203.0.113.203** and between **fc00:f853:ccd:e799::0** and **fc00:f853:ccd:e799::f**.

To explain the two BGP advertisements, consider an instance when MetalLB assigns the IP address of **203.0.113.200** to a service. With that IP address as an example, the speaker advertises the following two routes to BGP peers:

- **203.0.113.200/32**, with **localPref** set to **100** and the community set to the numeric value of the **NO_ADVERTISE** community. This specification indicates to the peer routers that they can use this route but they should not propagate information about this route to BGP peers.
- **203.0.113.200/30**, aggregates the load-balancer IP addresses assigned by MetalLB into a single route. MetalLB advertises the aggregated route to BGP peers with the community attribute set to **8000:800**. BGP peers propagate the **203.0.113.200/30** route to other BGP peers. When traffic is routed to a node with a speaker, the **203.0.113.200/32** route is used to forward the traffic into the cluster and to a pod that is associated with the service.

As you add more services and MetalLB assigns more load-balancer IP addresses from the pool, peer routers receive one local route, **203.0.113.20x/32**, for each service, and the **203.0.113.200/30** aggregate route. Each service that you add generates the **/30** route, but MetalLB deduplicates the routes to one BGP advertisement before communicating with peer routers.

4.2.3.1. Advertising an advanced address pool configuration with BGP

To ensure application services are reachable from external network peers through specific routing paths, configure MetalLB to advertise an advanced address pool by using the BGP. Establishing this advertisement allows your cluster to precisely communicate routing information to the external infrastructure.

The procedure demonstrates how to configure MetalLB to advertise an advanced address pool by using the BGP.

Prerequisites

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

Procedure

1. Create an IP address pool.

   a. Create a file, such as **ipaddresspool.yaml**, with content like the following example:

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  namespace: metallb-system
  name: doc-example-bgp-adv
  labels:
    zone: east
spec:
  addresses:
    - 203.0.113.200/30
    - fc00:f853:ccd:e799::/124
  autoAssign: false
# ...
```

   b. Apply the configuration for the IP address pool:

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

2. Create a BGP advertisement.

   a. Create a file, such as **bgpadvertisement1.yaml**, with content like the following example:

```
apiVersion: metallb.io/v1beta1
kind: BGPAdvertisement
metadata:
  name: bgpadvertisement-adv-1
  namespace: metallb-system
spec:
  ipAddressPools:
    - doc-example-bgp-adv
  communities:
    - 65535:65282
  aggregationLength: 32
  localPref: 100
# ...
```

   b. Apply the configuration:

```
$ oc apply -f bgpadvertisement1.yaml
```

   c. Create a file, such as **bgpadvertisement2.yaml**, with content like the following example:

```
apiVersion: metallb.io/v1beta1
kind: BGPAdvertisement
metadata:
  name: bgpadvertisement-adv-2
  namespace: metallb-system
spec:
  ipAddressPools:
    - doc-example-bgp-adv
  communities:
```

```
    - 8000:800
  aggregationLength: 30
  aggregationLengthV6: 124
# ...
```

d. Apply the configuration:

```
$ oc apply -f bgpadvertisement2.yaml
```

4.2.4. Advertising an IP address pool from a subset of nodes

To restrict IP address advertisements to a specific set of nodes, such as a public-facing subnet, configure the **nodeSelector** parameter in the **BGPAdvertisement** custom resource (CR). When you configure these selectors, OpenShift Container Platform routes external traffic only through designated network interfaces for improved security and isolation.

Prerequisites

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

Procedure

1. Create an IP address pool by using a CR:

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  namespace: metallb-system
  name: pool1
spec:
  addresses:
    - 4.4.4.100-4.4.4.200
    - 2001:100:4::200-2001:100:4::400
# ...
```

2. Control which cluster nodes advertise the IP address from **pool1** by setting the **.spec.nodeSelector** value in the **BGPAdvertisement** CR. The following example advertises the IP address from **pool1** only from **NodeA** and **NodeB**.

```
apiVersion: metallb.io/v1beta1
kind: BGPAdvertisement
metadata:
  name: example
spec:
  ipAddressPools:
  - pool1
  nodeSelector:
  - matchLabels:
      kubernetes.io/hostname: NodeA
  - matchLabels:
      kubernetes.io/hostname: NodeB
# ...
```

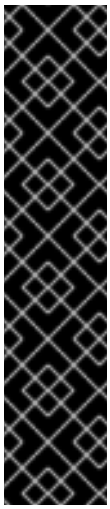
-

4.2.5. About the L2Advertisement custom resource

To configure how application services are announced over a Layer 2 network, define the properties in the **L2Advertisement** custom resource (CR). Establishing these parameters ensures that MetalLB correctly manages routing for your load-balancer IP addresses within the local network infrastructure

The following table details parameters for the **l2Advertisements** CR:

Table 4.4. L2 advertisements configuration

Parameter	Type	Description
metadata.name	**string**	Specifies the name for the L2 advertisement.
metadata.namespace	**string**	Specifies the namespace for the L2 advertisement. Specify the same namespace that the MetalLB Operator uses.
spec.ipAddressPools	**string**	Optional: The list of **IPAddressPools** to advertise with this advertisement, selected by name.
spec.ipAddressPoolSelectors	**string**	Optional: A selector for the **IPAddressPools** to advertise with this advertisement. This is for associating the **IPAddressPool** to the advertisement based on the label assigned to the **IPAddressPool** instead of the name itself. If no **IPAddressPool** is selected by this or by the list, the advertisement is applied to all the **IPAddressPools**.
spec.nodeSelectors	**string**	Optional: **NodeSelectors** limits the nodes to announce as next hops for the load balancer IP. If empty, MetalLB announces all nodes as next hops.  IMPORTANT Limiting the nodes to announce as next hops 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
spec.interfaces	**string**	Optional: The list of **interfaces** to announce the load balancer IP address.

4.2.6. Configuring MetalLB with an L2 advertisement

To provide fault-tolerant external IP addresses for cluster services, configure MetalLB to advertise an **IPAddressPool** by using the Layer 2 protocol. Establishing this advertisement ensures that application workloads remain reachable within the local network through standard address discovery protocols.

Prerequisites

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

Procedure

1. Create an IP address pool.
   a. Create a file, such as **ipaddresspool.yaml**, with content like the following example:

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  namespace: metallb-system
  name: doc-example-l2
spec:
  addresses:
    - 4.4.4.0/24
  autoAssign: false
# ...
```

   b. Apply the configuration for the IP address pool:

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

2. Create an L2 advertisement.
   a. Create a file, such as **l2advertisement.yaml**, with content like the following example:

```
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: l2advertisement
  namespace: metallb-system
spec:
  ipAddressPools:
   - doc-example-l2
  # ...
```

   b. Apply the configuration:

```
$ oc apply -f l2advertisement.yaml
```

4.2.7. Configuring MetalLB with an L2 advertisement and labels

To dynamically associate IP address pools with advertisements by using labels instead of specific names,

configure the **ipAddressPoolSelectors** parameter in the MetalLB custom resource (CR). By using selectors, you can manage network routing more efficiently by automatically grouping address pools as your cluster scales.

The example in the procedure shows how to configure MetalLB so that the **IPAddressPool** is advertised with the L2 protocol by configuring the **ipAddressPoolSelectors** field.

Prerequisites

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

Procedure

1. Create an IP address pool.
   a. Create a file, such as **ipaddresspool.yaml**, with content like the following example:

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  namespace: metallb-system
  name: doc-example-l2-label
  labels:
    zone: east
spec:
  addresses:
    - 172.31.249.87/32
# ...
```

   b. Apply the configuration for the IP address pool:

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

2. Create an L2 advertisement that advertises the IP address by using **ipAddressPoolSelectors**.
   a. Create a file, such as **l2advertisement.yaml**, with content like the following example:

```
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: l2advertisement-label
  namespace: metallb-system
spec:
  ipAddressPoolSelectors:
    - matchExpressions:
        - key: zone
          operator: In
          values:
            - east
# ...
```

   b. Apply the configuration:

```
$ oc apply -f l2advertisement.yaml
```

4.2.8. Configuring MetalLB with an L2 advertisement for selected interfaces

To restrict which network interfaces advertise assigned service IP addresses, configure the interfaces parameter in the MetalLB **L2Advertisement** custom resource (CR). Defining specific interfaces ensures that cluster services are reachable only through designated network paths for improved traffic management and isolation.

The example in the procedure shows how to configure MetalLB so that the IP address pool is advertised only from the network interfaces listed in the **interfaces** parameter of all nodes.

Prerequisites

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

Procedure

1. Create an IP address pool.
   a. Create a file, such as **ipaddresspool.yaml**, and enter the configuration details as shown in the following example:

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  namespace: metallb-system
  name: doc-example-l2
spec:
  addresses:
    - 4.4.4.0/24
  autoAssign: false
# ...
```

   b. Apply the configuration for the IP address pool as shown in the following example:

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

2. Create an L2 advertisement with the **interfaces** selector to advertise the IP address.
   a. Create a YAML file, such as **l2advertisement.yaml**, and enter the configuration details as shown the following example:

```
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: l2advertisement
  namespace: metallb-system
spec:
  ipAddressPools:
   - doc-example-l2
   interfaces:
```

```
    - interfaceA
    - interfaceB
# ...
```

b. Apply the configuration for the advertisement as shown in the following example:

```
$ oc apply -f l2advertisement.yaml
```

IMPORTANT

The interface selector does not affect how MetalLB chooses the node to announce a given IP by using L2. The chosen node does not announce the service if the node does not have the selected interface.

4.2.9. Configuring MetalLB with secondary networks

To enable traffic forwarding for MetalLB on a secondary network interface, add a machine configuration that allows IP packet forwarding between interfaces. Implementing this configuration ensures that application services remain reachable when they use nondefault network paths.

NOTE

From OpenShift Container Platform 4.14 the default network behavior does not allow forwarding of IP packets between network interfaces.

OpenShift Container Platform clusters upgraded from 4.13 are not affected because a global parameter is set during an upgrade to enable global IP forwarding.

To enable IP forwarding for the secondary interface, you have two options:

- Enable IP forwarding for a specific interface.
- Enable IP forwarding for all interfaces.

NOTE

Enabling IP forwarding for a specific interface provides more granular control, while enabling it for all interfaces applies a global setting.

Procedure

1. Patch the Cluster Network Operator, setting the parameter **routingViaHost** to **true** by running the following command:

```
$ oc patch network.operator cluster -p '{"spec":{"defaultNetwork":{"ovnKubernetesConfig":
{"gatewayConfig": {"routingViaHost": true} }}}}' --type=merge
```

2. Enable forwarding for a specific secondary interface, such as **bridge-net**, by creating and applying a **MachineConfig** CR:

   a. Base64-encode the string that is used to configure network kernel parameters by running the following command on your local machine:

```
$ echo -e "net.ipv4.conf.bridge-net.forwarding = 1" | base64 -w0
```

Example output

```
bmV0LmlwdjQuY29uZi5icmlkZ2UtbmV0LmZvcndhcmRpbmcgPSAxCg==
```

b. Create the **MachineConfig** CR to enable IP forwarding for the specified secondary interface named **bridge-net**.

c. Save the following YAML in the **enable-ip-forward.yaml** file:

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: <node_role>
  name: 81-enable-global-forwarding
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
      files:
      - contents:
          source: data:text/plain;charset=utf-
8;base64,bmV0LmlwdjQuY29uZi5icmlkZ2UtbmV0LmZvcndhcmRpbmcgPSAxCg==
          verification: {}
        filesystem: root
        mode: 420
        path: /etc/sysctl.d/enable-global-forwarding.conf
  osImageURL: ""
# ...
```

where:

<node_role>

Node role where you want to enable IP forwarding, for example, **worker**.

contents.source

Populate with the generated Base64 string.

d. Apply the configuration by running the following command:

```
$ oc apply -f enable-ip-forward.yaml
```

3. Optional: Enable IP forwarding globally by running the following command:

```
$ oc patch network.operator cluster -p '{"spec":{"defaultNetwork":{"ovnKubernetesConfig":
{"gatewayConfig":{"ipForwarding": "Global"}}}}}' --type=merge
```

Verification

1. After you apply the machine config, verify the changes by completing the following steps:

a. Enter into a debug session on the target node by running the following command. This command instantiates a debug pod called **<node-name>-debug**.

```
$ oc debug node/<node-name>
```

b. Set **/host** as the root directory within the debug shell by running the following command. The debug pod mounts root file system of the host in **/host** within the pod. By changing the root directory to **/host**, you can run binaries contained in the executable paths of the host.

```
$ chroot /host
```

c. Verify that IP forwarding is enabled by running the following command:

```
$ cat /etc/sysctl.d/enable-global-forwarding.conf
```

Example output

```
net.ipv4.conf.bridge-net.forwarding = 1
```

The output indicates that IPv4 forwarding is enabled on the **bridge-net** interface.

4.2.10. Additional resources

- Configuring a community alias

4.3. CONFIGURING METALLB BGP PEERS

To establish BGP sessions and advertise routes for load balancer services, configure MetalLB Border Gateway Protocol (BGP) peer custom resources (CRs). Defining these peers ensures that your network infrastructure receives accurate routing information to reach cluster application workloads.

You can add, modify, and delete BGP peers. The MetalLB Operator uses the BGP peer custom resources to identify the peers that MetalLB **speaker** pods contact to start BGP sessions. The peers receive the route advertisements for the load-balancer IP addresses that MetalLB assigns to services.

4.3.1. About the BGP peer custom resource

To establish network connectivity and advertise routes for load-balancer services, configure the properties of the MetalLB Border Gateway Protocol (BGP) peer custom resource (CR). Establishing these parameters ensures that your cluster authorizes specific routing peers to direct external traffic correctly to your application workloads.

The following table describes the parameters for the BGP peer CR:

Table 4.5. MetalLB BGP peer custom resource

Parameter	Type	Description
metadata.name	**string**	Specifies the name for the BGP peer CR.

Parameter	Type	Description
metadata.namespace	**string**	Specifies the namespace for the BGP peer CR.
spec.myASN	**integer**	Specifies the Autonomous System Number (ASN) for the local end of the BGP session. In all BGP peer CRs that you add, specify the same value. The range is **0** to **4294967295**.
spec.peerASN	**integer**	Specifies the ASN for the remote end of the BGP session. The range is **0** to **4294967295**. If you use this parameter, you cannot specify a value in the **spec.dynamicASN** parameter.
spec.dynamicASN	**string**	Detects the ASN to use for the remote end of the session without explicitly setting it. Specify **internal** for a peer with the same ASN, or **external** for a peer with a different ASN. If you use this parameter, you cannot specify a value in the **spec.peerASN** parameter.
spec.peerAddress	**string**	Specifies the IP address of the peer to contact for establishing the BGP session.
spec.sourceAddress	**string**	Optional: Specifies the IP address to use when establishing the BGP session. The value must be an IPv4 address.
spec.peerPort	**integer**	Optional: Specifies the network port of the peer to contact for establishing the BGP session. The range is **0** to **16384**.
spec.holdTime	**string**	Optional: Specifies the duration for the hold time to propose to the BGP peer. The minimum value is 3 seconds (**3s**). The common units are seconds and minutes, such as **3s**, **1m**, and **5m30s**. To detect path failures more quickly, also configure BFD.
spec.keepaliveTime	**string**	Optional: Specifies the maximum interval between sending keep-alive messages to the BGP peer. If you specify this parameter, you must also specify a value for the **holdTime** parameter. The specified value must be less than the value for the **holdTime** parameter.
spec.routerID	**string**	Optional: Specifies the router ID to advertise to the BGP peer. If you specify this parameter, you must specify the same value in every BGP peer custom resource that you add.
spec.password	**string**	Optional: Specifies the MD5 password to send to the peer for routers that enforce TCP MD5 authenticated BGP sessions.

Parameter	Type	Description
spec.passwordSecret	**string**	Optional: Specifies name of the authentication secret for the BGP peer. The secret must live in the **metallb** namespace and be of type basic-auth.
spec.bfdProfile	**string**	Optional: Specifies the name of a BFD profile.
spec.nodeSelectors	**object[]**	Optional: Specifies a selector, using match expressions and match labels, to control which nodes can connect to the BGP peer.
spec.ebgpMultiHop	**boolean**	Optional: Specifies that the BGP peer is multiple network hops away. If the BGP peer is not directly connected to the same network, the speaker cannot establish a BGP session unless this parameter is set to **true**. This parameter applies to*external BGP*.External BGP is the term that is used to describe when a BGP peer belongs to a different Autonomous System.
connectTime	**duration**	Specifies how long BGP waits between connection attempts to a neighbor.

NOTE

The **passwordSecret** parameter is mutually exclusive with the **password** parameter, and contains a reference to a secret containing the password to use. Setting both parameters results in a failure of the parsing.

4.3.2. Configuring a BGP peer

To exchange routing information and advertise IP addresses for load balancer services, configure MetalLB BGP peer CRs. Establishing these peers ensures that your network infrastructure can reach and correctly route traffic to cluster application workloads.

You can add a BGP peer custom resource to exchange routing information with network routers and advertise the IP addresses for services.

Prerequisites

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

Procedure

1. Create a file, such as **bgppeer.yaml**, with content like the following example:

```
apiVersion: metallb.io/v1beta2
kind: BGPPeer
metadata:
  namespace: metallb-system
  name: doc-example-peer
spec:
  peerAddress: 10.0.0.1
  peerASN: 64501
  myASN: 64500
  routerID: 10.10.10.10
# ...
```

2. Apply the BGP peer configuration by entering the following command:

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

4.3.3. Configure a specific set of BGP peers for a given address pool

To assign specific IP address pools to designated BGP peers, configure MetalLB BGP advertisements. Establishing these mappings ensures that your cluster advertises designated network ranges only to authorized routing peers for precise external traffic control.

This procedure demonstrates the following tasks:

- Configure a set of address pools: **pool1** and **pool2**.
- Configure a set of BGP peers: **peer1** and **peer2**.
- Configure BGP advertisement to assign **pool1** to **peer1** and **pool2** to **peer2**.

Prerequisites

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

Procedure

1. Create address pool **pool1**.
   a. Create a file, such as **ipaddresspool1.yaml**, with content like the following example:

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  namespace: metallb-system
  name: pool1
spec:
  addresses:
    - 4.4.4.100-4.4.4.200
    - 2001:100:4::200-2001:100:4::400
# ...
```

   b. Apply the configuration for the IP address pool **pool1**:

```
$ oc apply -f ipaddresspool1.yaml
```

2. Create address pool **pool2**.

   a. Create a file, such as **ipaddresspool2.yaml**, with content like the following example:

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  namespace: metallb-system
  name: pool2
spec:
  addresses:
    - 5.5.5.100-5.5.5.200
    - 2001:100:5::200-2001:100:5::400
# ...
```

   b. Apply the configuration for the IP address pool **pool2**:

```
$ oc apply -f ipaddresspool2.yaml
```

3. Create BGP **peer1**.

   a. Create a file, such as **bgppeer1.yaml**, with content like the following example:

```
apiVersion: metallb.io/v1beta2
kind: BGPPeer
metadata:
  namespace: metallb-system
  name: peer1
spec:
  peerAddress: 10.0.0.1
  peerASN: 64501
  myASN: 64500
  routerID: 10.10.10.10
# ...
```

   b. Apply the configuration for the BGP **peer1**:

```
$ oc apply -f bgppeer1.yaml
```

4. Create BGP **peer2**.

   a. Create a file, such as **bgppeer2.yaml**, with content like the following example:

```
apiVersion: metallb.io/v1beta2
kind: BGPPeer
metadata:
  namespace: metallb-system
  name: peer2
spec:
  peerAddress: 10.0.0.2
  peerASN: 64501
```

```
  myASN: 64500
  routerID: 10.10.10.10
# ...
```

b. Apply the configuration for the BGP **peer2**:

```
$ oc apply -f bgppeer2.yaml
```

5. Create BGP advertisement 1.

a. Create a file, such as **bgpadvertisement1.yaml**, with content like the following example:

```
apiVersion: metallb.io/v1beta1
kind: BGPAdvertisement
metadata:
  name: bgpadvertisement-1
  namespace: metallb-system
spec:
  ipAddressPools:
    - pool1
  peers:
    - peer1
  communities:
    - 65535:65282
  aggregationLength: 32
  aggregationLengthV6: 128
  localPref: 100
# ...
```

b. Apply the configuration:

```
$ oc apply -f bgpadvertisement1.yaml
```

6. Create BGP advertisement 2.

a. Create a file, such as **bgpadvertisement2.yaml**, with content like the following example:

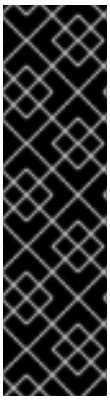
```
apiVersion: metallb.io/v1beta1
kind: BGPAdvertisement
metadata:
  name: bgpadvertisement-2
  namespace: metallb-system
spec:
  ipAddressPools:
    - pool2
  peers:
    - peer2
  communities:
    - 65535:65282
  aggregationLength: 32
  aggregationLengthV6: 128
  localPref: 100
# ...
```

b. Apply the configuration:

```
$ oc apply -f bgpadvertisement2.yaml
```

4.3.4. Exposing a service through a network VRF

To isolate network traffic and manage multiple routing tables, expose a service through a virtual routing and forwarding (VRF) instance. Associating a VRF with a MetalLB BGP peer ensures that external traffic is segmented and correctly routed to the intended application workloads.

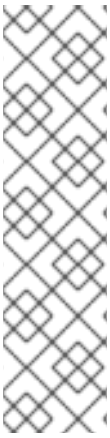


IMPORTANT

Exposing a service through a VRF on a BGP peer 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 .

By using a VRF on a network interface to expose a service through a BGP peer, you can segregate traffic to the service, configure independent routing decisions, and enable multi-tenancy support on a network interface.



NOTE

By establishing a BGP session through an interface belonging to a network VRF, MetalLB can advertise services through that interface and enable external traffic to reach the service through this interface. However, the network VRF routing table is different from the default VRF routing table used by OVN-Kubernetes. Therefore, the traffic cannot reach the OVN-Kubernetes network infrastructure.

To enable the traffic directed to the service to reach the OVN-Kubernetes network infrastructure, you must configure routing rules to define the next hops for network traffic. See the **NodeNetworkConfigurationPolicy** resource in "Managing symmetric routing with MetalLB" in the *Additional resources* section for more information.

The following high-level steps demonstrate how to expose a service through a network VRF with a BGP peer:

1. Define a BGP peer and add a network VRF instance.
2. Specify an IP address pool for MetalLB.
3. Configure a BGP route advertisement for MetalLB to advertise a route by using the specified IP address pool and the BGP peer associated with the VRF instance.
4. Deploy a service to test the configuration.

Prerequisites

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

- You logged in as a user with **cluster-admin** privileges.
- You defined a **NodeNetworkConfigurationPolicy** (NNCP) to associate a Virtual Routing and Forwarding (VRF) instance with a network interface. For more information about completing this prerequisite, see the *Additional resources* section.
- You installed MetalLB on your cluster.

Procedure

1. Create a **BGPPeer** custom resource (CR):
   a. Create a file, such as **frrviavrf.yaml**, with content like the following example:

```
apiVersion: metallb.io/v1beta2
kind: BGPPeer
metadata:
  name: frrviavrf
  namespace: metallb-system
spec:
  myASN: 100
  peerASN: 200
  peerAddress: 192.168.130.1
  vrf: ens4vrf
# ...
```

   spec.vrf: Specifies the network VRF instance to associate with the BGP peer. MetalLB can advertise services and make routing decisions based on the routing information in the VRF.

NOTE

You must configure this network VRF instance in a **NodeNetworkConfigurationPolicy** CR. See the *Additional resources* for more information.

   b. Apply the configuration for the BGP peer by running the following command:

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

2. Create an **IPAddressPool** CR:
   a. Create a file, such as **first-pool.yaml**, with content like the following example:

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: first-pool
  namespace: metallb-system
spec:
  addresses:
  - 192.169.10.0/32
# ...
```

   b. Apply the configuration for the IP address pool by running the following command:

```
$ oc apply -f first-pool.yaml
```

3. Create a **BGPAdvertisement** CR:

   a. Create a file, such as **first-adv.yaml**, with content like the following example:

```
apiVersion: metallb.io/v1beta1
kind: BGPAdvertisement
metadata:
  name: first-adv
  namespace: metallb-system
spec:
  ipAddressPools:
    - first-pool
  peers:
    - frrviavrf
# ...
```

   peers.frrviavrf: In this example, MetalLB advertises a range of IP addresses from the **first-pool** IP address pool to the **frrviavrf** BGP peer.

   b. Apply the configuration for the BGP advertisement by running the following command:

```
$ oc apply -f first-adv.yaml
```

4. Create a **Namespace**, **Deployment**, and **Service** CR:

   a. Create a file, such as **deploy-service.yaml**, with content like the following example:

```
apiVersion: v1
kind: Namespace
metadata:
  name: test
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: server
  namespace: test
spec:
  selector:
    matchLabels:
      app: server
  template:
    metadata:
      labels:
        app: server
    spec:
      containers:
      - name: server
        image: nginx
        ports:
        - name: http
          containerPort: 80
---
```

```
apiVersion: v1
kind: Service
metadata:
  name: server1
  namespace: test
spec:
  ports:
  - name: http
    port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: server
  type: LoadBalancer
# ...
```

b. Apply the configuration for the namespace, deployment, and service by running the following command:

```
$ oc apply -f deploy-service.yaml
```

Verification

1. Identify a MetalLB speaker pod by running the following command:

```
$ oc get -n metallb-system pods -l component=speaker
```

Example output

```
NAME          READY   STATUS    RESTARTS   AGE
speaker-c6c5f   6/6     Running   0          69m
```

2. Verify that the state of the BGP session is **Established** in the speaker pod by running the following command, replacing the variables to match your configuration:

```
$ oc exec -n metallb-system <speaker_pod> -c frr -- vtysh -c "show bgp vrf <vrf_name>
neigh"
```

Example output

```
BGP neighbor is 192.168.30.1, remote AS 200, local AS 100, external link
  BGP version 4, remote router ID 192.168.30.1, local router ID 192.168.30.71
  BGP state = Established, up for 04:20:09

...
```

3. Verify that the service is advertised correctly by running the following command:

```
$ oc exec -n metallb-system <speaker_pod> -c frr -- vtysh -c "show bgp vrf <vrf_name> ipv4"
```

4.3.5. Additional resources

- About virtual routing and forwarding
- Example: Network interface with a VRF instance node network configuration policy
- Configuring an egress service
- Managing symmetric routing with MetalLB

4.3.6. Example BGP peer configurations

To precisely manage network topology and improve routing resilience, configure MetalLB BGP peer settings that limit node connectivity and incorporate BFD profiles. Defining these parameters ensures that your cluster maintains secure peer relationships and rapidly detects path failures for high service availability.

Example of limiting which nodes connect to a BGP peer

You can specify the **nodeSelectors** parameter to control which nodes can connect to a BGP peer.

```
apiVersion: metallb.io/v1beta2
kind: BGPPeer
metadata:
  name: doc-example-nodesel
  namespace: metallb-system
spec:
  peerAddress: 10.0.20.1
  peerASN: 64501
  myASN: 64500
  nodeSelectors:
  - matchExpressions:
    - key: kubernetes.io/hostname
      operator: In
      values: [compute-1.example.com, compute-2.example.com]
# ...
```

Example of specifying a BFD profile for a BGP peer

You can specify a BFD profile to associate with BGP peers. BFD complements BGP by providing faster detection of communication failures between peers than BGP alone.

```
apiVersion: metallb.io/v1beta2
kind: BGPPeer
metadata:
  name: doc-example-peer-bfd
  namespace: metallb-system
spec:
  peerAddress: 10.0.20.1
  peerASN: 64501
  myASN: 64500
  holdTime: "10s"
  bfdProfile: doc-example-bfd-profile-full
# ...
```

NOTE

Deleting the bidirectional forwarding detection (BFD) profile and removing the **bfdProfile** added to the border gateway protocol (BGP) peer resource does not disable the BFD. Instead, the BGP peer starts using the default BFD profile. To disable BFD from a BGP peer resource, delete the BGP peer configuration and recreate it without a BFD profile. For more information, see BZ#2050824.

Example of specifying BGP peers for dual-stack networking

To support dual-stack networking, add one BGP peer custom resource for IPv4 and one BGP peer custom resource for IPv6.

```
apiVersion: metallb.io/v1beta2
kind: BGPPeer
metadata:
  name: doc-example-dual-stack-ipv4
  namespace: metallb-system
spec:
  peerAddress: 10.0.20.1
  peerASN: 64500
  myASN: 64500
---
apiVersion: metallb.io/v1beta2
kind: BGPPeer
metadata:
  name: doc-example-dual-stack-ipv6
  namespace: metallb-system
spec:
  peerAddress: 2620:52:0:88::104
  peerASN: 64500
  myASN: 64500
# ...
```

4.3.7. Additional resources

- Configuring services to use MetalLB

4.4. CONFIGURING COMMUNITY ALIAS

As a cluster administrator, you can configure a community alias and use it across different advertisements.

4.4.1. About the community custom resource

To simplify BGP configuration, define named aliases for community values by using the community custom resource. You can reference these aliases when advertising **ipAddressPools** with the **BGPAdvertisement** resource.

The fields for the **community** custom resource are described in the following table.

NOTE

The **community** CRD applies only to BGPAdvertisement.

Table 4.6. MetalLB community custom resource

Field	Type	Description
metadata.name	**string**	Specifies the name for the **community**.
metadata.namespace	**string**	Specifies the namespace for the **community**. Specify the same namespace that the MetalLB Operator uses.
spec.communities	**string**	Specifies a list of BGP community aliases that can be used in BGPAdvertisements. A community alias consists of a pair of name (alias) and value (number:number). Link the BGPAdvertisement to a community alias by referring to the alias name in its **spec.communities** field.

Table 4.7. CommunityAlias

Field	Type	Description
name	**string**	The name of the alias for the **community**.
value	**string**	The BGP **community** value corresponding to the given name.

4.4.2. Configuring MetalLB with a BGP advertisement and community alias

To advertise an **IPAddressPool** by using the BGP protocol, configure MetalLB with a community alias. This configuration sets the alias to the numeric value of the **NO_ADVERTISE** community.

In the following example, the peer BGP router **doc-example-peer-community** receives one **203.0.113.200/32** route and one **fc00:f853:ccd:e799::1/128** route for each load-balancer IP address that MetalLB assigns to a service. A community alias is configured with the **NO_ADVERTISE** community.

Prerequisites

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

Procedure

1. Create an IP address pool.
   a. Create a file, such as **ipaddresspool.yaml**, with content like the following example:

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  namespace: metallb-system
  name: doc-example-bgp-community
spec:
```

```
    addresses:
      - 203.0.113.200/30
      - fc00:f853:ccd:e799::/124
```

b. Apply the configuration for the IP address pool:

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

2. Create a community alias named **community1**.

```
apiVersion: metallb.io/v1beta1
kind: Community
metadata:
  name: community1
  namespace: metallb-system
spec:
  communities:
    - name: NO_ADVERTISE
      value: '65535:65282'
```

3. Create a BGP peer named **doc-example-bgp-peer**.

a. Create a file, such as **bgppeer.yaml**, with content like the following example:

```
apiVersion: metallb.io/v1beta2
kind: BGPPeer
metadata:
  namespace: metallb-system
  name: doc-example-bgp-peer
spec:
  peerAddress: 10.0.0.1
  peerASN: 64501
  myASN: 64500
  routerID: 10.10.10.10
```

b. Apply the configuration for the BGP peer:

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

4. Create a BGP advertisement with the community alias.

a. Create a file, such as **bgpadvertisement.yaml**, with content like the following example:

```
apiVersion: metallb.io/v1beta1
kind: BGPAdvertisement
metadata:
  name: bgp-community-sample
  namespace: metallb-system
spec:
  aggregationLength: 32
  aggregationLengthV6: 128
  communities:
    - NO_ADVERTISE (1)
  ipAddressPools:
```

```
    - doc-example-bgp-community
  peers:
    - doc-example-peer
```

where:

NO_ADVERTISE: Specifies the **CommunityAlias.name** here and not the community custom resource (CR) name.

b. Apply the configuration:

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

4.5. CONFIGURING METALLB BFD PROFILES

To achieve faster path failure detection for BGP sessions, configure MetalLB Bidirectional Forwarding Detection (BFD) profiles. Establishing these profiles ensures that your network routing remains highly available and responsive by identifying connectivity issues more quickly than standard protocols.

You can add, modify, and delete BFD profiles. The MetalLB Operator uses the BFD profile custom resources (CRs) to identify the BGP sessions that use BFD.

4.5.1. About the BFD profile custom resource

To enable rapid detection of communication failures between routing peers, configure the properties of the MetalLB BFD profile custom resource (CR). Defining these parameters ensures that network traffic is quickly rerouted if a path becomes unavailable, maintaining high cluster reachability and stability.

The following table describes parameters for the BFD profile CR:

Table 4.8. BFD profile custom resource

Parameter	Type	Description
metadata.name	**string**	Specifies the name for the BFD profile custom resource.
metadata.namespace	**string**	Specifies the namespace for the BFD profile custom resource.
spec.detectMultiplier	**integer**	Specifies the detection multiplier to determine packet loss. The remote transmission interval is multiplied by this value to determine the connection loss detection timer. For example, when the local system has the detect multiplier set to **3** and the remote system has the transmission interval set to **300**, the local system detects failures only after **900** ms without receiving packets. The range is **2** to **255**. The default value is **3**.

Parameter	Type	Description
spec.echoMode	**boolean**	Specifies the echo transmission mode. If you are not using distributed BFD, echo transmission mode works only when the peer is also FRR. The default value is **false** and echo transmission mode is disabled. When echo transmission mode is enabled, consider increasing the transmission interval of control packets to reduce bandwidth usage. For example, consider increasing the transmit interval to **2000** ms.
spec.echoInterval	**integer**	Specifies the minimum transmission interval, less jitter, that this system uses to send and receive echo packets. The range is **10** to **60000**. The default value is **50** ms.
spec.minimumTtl	**integer**	Specifies the minimum expected TTL for an incoming control packet. This field applies to multi-hop sessions only. The purpose of setting a minimum TTL is to make the packet validation requirements more stringent and avoid receiving control packets from other sessions. The default value is **254** and indicates that the system expects only one hop between this system and the peer.
spec.passiveMode	**boolean**	Specifies whether a session is marked as active or passive. A passive session does not attempt to start the connection. Instead, a passive session waits for control packets from a peer before it begins to reply. Marking a session as passive is useful when you have a router that acts as the central node of a star network and you want to avoid sending control packets that you do not need the system to send. The default value is **false** and marks the session as active.
spec.receiveInterval	**integer**	Specifies the minimum interval that this system is capable of receiving control packets. The range is **10** to **60000**. The default value is **300** ms.
spec.transmitInterval	**integer**	Specifies the minimum transmission interval, less jitter, that this system uses to send control packets. The range is **10** to **60000**. The default value is **300** ms.

4.5.2. Configuring a BFD profile

To achieve faster path failure detection for BGP sessions, configure a MetalLB BFD profile and associate it with a BGP peer. Establishing these profiles ensures that your network routing remains highly available and responsive by identifying connectivity issues more rapidly than standard protocols.

Prerequisites

Prerequisites

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

Procedure

1. Create a file, such as **bfdprofile.yaml**, with content like the following example:

```
apiVersion: metallb.io/v1beta1
kind: BFDProfile
metadata:
  name: doc-example-bfd-profile-full
  namespace: metallb-system
spec:
  receiveInterval: 300
  transmitInterval: 300
  detectMultiplier: 3
  echoMode: false
  passiveMode: true
  minimumTtl: 254
# ...
```

2. Apply the configuration for the BFD profile:

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

4.5.3. Additional resources

- Configuring MetalLB BGP peers

4.6. CONFIGURING SERVICES TO USE METALLB

To ensure predictable network endpoints, control how MetalLB assigns IP addresses to services of type **LoadBalancer**. Requesting specific addresses or pools ensures that your applications receive valid IP assignments that align with your specific network addressing plan.

4.6.1. Request a specific IP address

To assign a specific, static IP address to a service, configure the **spec.loadBalancerIP** parameter in the service specification.

MetalLB attempts to assign the requested address from the configured address pools, ensuring that your service is reachable at a designated, static network endpoint. If the requested IP address is not within any range, MetalLB reports a warning.

Example service YAML for a specific IP address

```
apiVersion: v1
kind: Service
metadata:
  name: <service_name>
  annotations:
```

```
    metallb.io/address-pool: <address_pool_name>
spec:
  selector:
    <label_key>: <label_value>
  ports:
    - port: 8080
      targetPort: 8080
      protocol: TCP
  type: LoadBalancer
  loadBalancerIP: <ip_address>
```

If MetalLB cannot assign the requested IP address, the **EXTERNAL-IP** for the service reports **<pending>** and running **oc describe service <service_name>** includes an event like the following example:

Example event when MetalLB cannot assign a requested IP address

```
  ...
Events:
  Type     Reason          Age    From              Message
  ----     ------          ----   ----              -------
  Warning  AllocationFailed  3m16s  metallb-controller  Failed to allocate IP for "default/invalid-request": "4.3.2.1" is not allowed in config
```

4.6.2. Request an IP address from a specific pool

To ensure predictable network endpoints, control how MetalLB assigns IP addresses to services of type **LoadBalancer**. Requesting specific addresses or pools ensures that your applications receive valid IP assignments that align with your specific network addressing plan.

Example service YAML for an IP address from a specific pool

```
apiVersion: v1
kind: Service
metadata:
  name: <service_name>
  annotations:
    metallb.io/address-pool: <address_pool_name>
spec:
  selector:
    <label_key>: <label_value>
  ports:
    - port: 8080
      targetPort: 8080
      protocol: TCP
  type: LoadBalancer
```

If the address pool that you specify for **<address_pool_name>** does not exist, MetalLB attempts to assign an IP address from any pool that permits automatic assignment.

4.6.3. Accept any IP address

To automatically allocate IP addresses to services without manual specification, configure MetalLB address pools to permit automatic assignment. MetalLB dynamically assigns available addresses from these pools, ensuring seamless service deployment and network connectivity.

Example service YAML for accepting any IP address

```
apiVersion: v1
kind: Service
metadata:
  name: <service_name>
spec:
  selector:
    <label_key>: <label_value>
  ports:
    - port: 8080
      targetPort: 8080
      protocol: TCP
  type: LoadBalancer
```

4.6.4. Share a specific IP address

To colocate multiple services on a single IP address, apply the **metallb.io/allow-shared-ip** annotation to your service specifications. Configuring this annotation authorizes MetalLB to assign the same IP to multiple services, enabling efficient address usage while distinguishing traffic by port.

By default, services do not share IP addresses.

```
apiVersion: v1
kind: Service
metadata:
  name: service-http
  annotations:
    metallb.io/address-pool: doc-example
    metallb.io/allow-shared-ip: "web-server-svc"
spec:
  ports:
    - name: http
      port: 80
      protocol: TCP
      targetPort: 8080
  selector:
    <label_key>: <label_value>
  type: LoadBalancer
  loadBalancerIP: 172.31.249.7
# ...
---
apiVersion: v1
kind: Service
metadata:
  name: service-https
  annotations:
    metallb.io/address-pool: doc-example
    metallb.io/allow-shared-ip: "web-server-svc"
spec:
  ports:
```

```
    - name: https
      port: 443
      protocol: TCP
      targetPort: 8080
  selector:
    <label_key>: <label_value>
  type: LoadBalancer
  loadBalancerIP: 172.31.249.7
# ...
```

where:

metallb.io/allow-shared-ip

Specifies the same value for the **metallb.io/allow-shared-ip** annotation. This value is referred to as the *sharing key*.

name.port

Specifies different port numbers for the services.

spec.selector

Specifies identical pod selectors if you must specify **externalTrafficPolicy: local** so the services send traffic to the same set of pods. If you use the **cluster** external traffic policy, then the pod selectors do not need to be identical.

spec.loadBalancerIP

Optional parameter. If you specify the three preceding items, MetalLB might colocate the services on the same IP address. To ensure that services share an IP address, specify the IP address to share.

By default, Kubernetes does not allow multiprotocol load balancer services. This limitation would normally make it impossible to run a service like DNS that needs to listen on both TCP and UDP. To work around this limitation of Kubernetes with MetalLB, create two services:

- For one service, specify TCP and for the second service, specify UDP.
- In both services, specify the same pod selector.
- Specify the same sharing key and **spec.loadBalancerIP** value to colocate the TCP and UDP services on the same IP address.

4.6.5. Configuring a service with MetalLB

To expose an application to external network traffic, configure a load-balancing service. MetalLB assigns an external IP address from a configured address pool, ensuring that your application is reachable from outside the cluster.

Prerequisites

- Install the OpenShift CLI (**oc**).
- Install the MetalLB Operator and start MetalLB.
- Configure at least one address pool.
- Configure your network to route traffic from the clients to the host network for the cluster.

Procedure

1. Create a **<service_name>.yaml** file. In the file, set the **spec.type** parameter to **LoadBalancer**. Refer to the examples for information about how to request the external IP address that MetalLB assigns to the service.

2. Create the service:

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

Example output

```
service/<service_name> created
```

Verification

- Describe the service:

```
$ oc describe service <service_name>
```

Example output

```
Name:                     <service_name>
Namespace:                default
Labels:                   <none>
Annotations:              metallb.io/address-pool: doc-example
Selector:                 app=service_name
Type:                     LoadBalancer
IP Family Policy:         SingleStack
IP Families:              IPv4
IP:                       10.105.237.254
IPs:                      10.105.237.254
LoadBalancer Ingress:     192.168.100.5
Port:                     <unset>  80/TCP
TargetPort:               8080/TCP
NodePort:                 <unset>  30550/TCP
Endpoints:                10.244.0.50:8080
Session Affinity:         None
External Traffic Policy:  Cluster
Events:
  Type    Reason        Age               From            Message
  ----    ------        ----              ----            -------
  Normal  nodeAssigned  32m (x2 over 32m)  metallb-speaker  announcing from node "
<node_name>"
```

where:

Annotations

Specifies the annotation that is present if you request an IP address from a specific pool.

Type

Specifies the service type that must indicate **LoadBalancer**.

LoadBalancer Ingress

Specifies the indicates the external IP address if the service is assigned correctly.

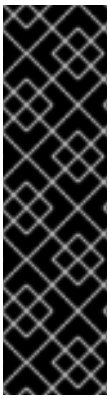
Events

Specifies the events parameter that indicates the node name that is assigned to announce the external IP address. If you experience an error, the events parameter indicates the reason for the error.

4.7. MANAGING SYMMETRIC ROUTING WITH METALLB

As a cluster administrator, you can effectively manage traffic for pods behind a MetalLB load-balancer service with multiple host interfaces by implementing features from MetalLB, NMState, and OVN-Kubernetes. By combining these features in this context, you can provide symmetric routing, traffic segregation, and support clients on different networks with overlapping CIDR addresses.

To achieve this functionality, learn how to implement virtual routing and forwarding (VRF) instances with MetalLB, and configure egress services.



IMPORTANT

Configuring symmetric traffic by using a VRF instance with MetalLB and an egress service 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 .

4.7.1. Challenges of managing symmetric routing with MetalLB

To resolve network isolation and asymmetric routing challenges on multiple host interfaces, implement a configuration combining MetalLB, NMState, and OVN-Kubernetes. This solution ensures symmetric routing and prevents overlapping CIDR addresses without requiring manual static route maintenance.

One option to ensure that return traffic reaches the correct client is to use static routes. However, with this solution, MetalLB cannot isolate the services and then announce each service through a different interface. Additionally, static routing requires manual configuration and requires maintenance if remote sites are added.

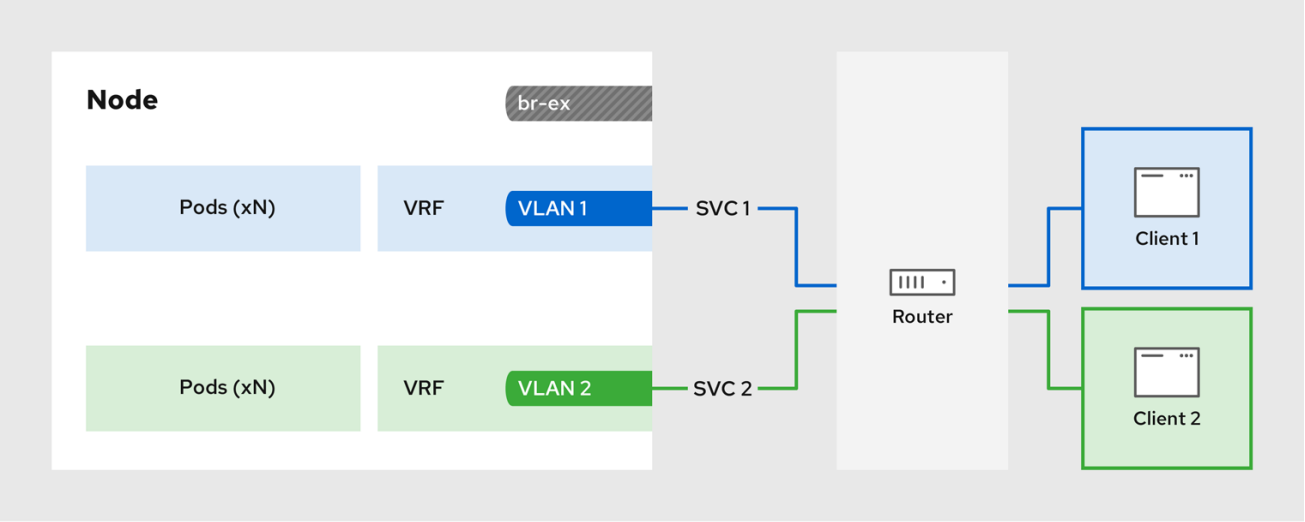
A further challenge of symmetric routing when implementing a MetalLB service is scenarios where external systems expect the source and destination IP address for an application to be the same. The default behavior for OpenShift Container Platform is to assign the IP address of the host network interface as the source IP address for traffic originating from pods. This is problematic with multiple host interfaces.

You can overcome these challenges by implementing a configuration that combines features from MetalLB, NMState, and OVN-Kubernetes.

4.7.2. Overview of managing symmetric routing by using VRFs with MetalLB

You can overcome the challenges of implementing symmetric routing by using NMState to configure a VRF instance on a host, associating the VRF instance with a MetalLB **BGPPeer** resource, and configuring an egress service for egress traffic with OVN-Kubernetes.

Figure 4.1. Network overview of managing symmetric routing by using VRFs with MetalLB



The configuration process involves three stages:

1: Define a VRF and routing rules

- Configure a **NodeNetworkConfigurationPolicy** custom resource (CR) to associate a VRF instance with a network interface.
- Use the VRF routing table to direct ingress and egress traffic.

2: Link the VRF to a MetalLB BGPPeer

- Configure a MetalLB **BGPPeer** resource to use the VRF instance on a network interface.
- By associating the **BGPPeer** resource with the VRF instance, the designated network interface becomes the primary interface for the BGP session, and MetalLB advertises the services through this interface.

3: Configure an egress service

- Configure an egress service to choose the network associated with the VRF instance for egress traffic.
- Optional: Configure an egress service to use the IP address of the MetalLB load-balancer service as the source IP for egress traffic.

4.7.3. Configuring symmetric routing by using VRFs with MetalLB

To ensure that applications behind a MetalLB service use the same network path for both ingress and egress, configure symmetric routing by using Virtual Routing and Forwarding (VRF).

The example in the procedure associates a VRF routing table with MetalLB and an egress service to enable symmetric routing for ingress and egress traffic for pods behind a **LoadBalancer** service.

NOTE

- If you use the **sourceIPBy: "LoadBalancerIP"** setting in the **EgressService** CR, you must specify the load-balancer node in the **BGPAdvertisement** custom resource (CR).
- You can use the **sourceIPBy: "Network"** setting on clusters that use OVN-Kubernetes configured with the **gatewayConfig.routingViaHost** specification set to **true** only. Additionally, if you use the **sourceIPBy: "Network"** setting, you must schedule the application workload on nodes configured with the network VRF instance.

Prerequisites

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

Procedure

1. Create a **NodeNetworkConfigurationPolicy** CR to define the VRF instance:
   a. Create a file, such as **node-network-vrf.yaml**, with content like the following example:

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: vrfpolicy
spec:
  nodeSelector:
    vrf: "true"
  maxUnavailable: 3
  desiredState:
    interfaces:
    - name: ens4vrf
      type: vrf
      state: up
      vrf:
        port:
        - ens4
        route-table-id: 2
    - name: ens4
      type: ethernet
      state: up
      ipv4:
        address:
        - ip: 192.168.130.130
          prefix-length: 24
        dhcp: false
        enabled: true
    routes:
      config:
```

```
      - destination: 0.0.0.0/0
        metric: 150
        next-hop-address: 192.168.130.1
        next-hop-interface: ens4
        table-id: 2
    route-rules:
      config:
      - ip-to: 172.30.0.0/16
        priority: 998
        route-table: 254
      - ip-to: 10.132.0.0/14
        priority: 998
        route-table: 254
      - ip-to: 169.254.0.0/17
        priority: 998
        route-table: 254
# ...
```

where:

metadata.name

Specifies the name of the policy.

nodeSelector.vrf

Specifies the policy for all nodes with the label **vrf:true**.

interfaces.name.ens4vrf

Specifies the name of the interface.

interfaces.type

Specifies the type of interface. This example creates a VRF instance.

vrf.port

Specifies the node interface that the VRF attaches to.

vrf.route-table-id

Specifies the name of the route table ID for the VRF.

`interfaces.name.ens4 `

Specifies the IPv4 address of the interface associated with the VRF.

routes

Specifies the configuration for network routes. The **next-hop-address** field defines the IP address of the next hop for the route. The **next-hop-interface** field defines the outgoing interface for the route. In this example, the VRF routing table is **2**, which references the ID that you define in the **EgressService** CR.

route-rules

Specifies additional route rules. The **ip-to** fields must match the **Cluster Network** CIDR, **Service Network** CIDR, and **Internal Masquerade** subnet CIDR. You can view the values for these CIDR address specifications by running the following command: **oc describe network.operator/cluster**.

route-rules.route-table

Specifies the main routing table that the Linux kernel uses when calculating routes has the ID **254**.

b. Apply the policy by running the following command:

```
$ oc apply -f node-network-vrf.yaml
```

2. Create a **BGPPeer** custom resource (CR):

a. Create a file, such as **frr-via-vrf.yaml**, with content like the following example:

```
apiVersion: metallb.io/v1beta2
kind: BGPPeer
metadata:
  name: frrviavrf
  namespace: metallb-system
spec:
  myASN: 100
  peerASN: 200
  peerAddress: 192.168.130.1
  vrf: ens4vrf
# ...
```

where:

spec.vrf

Specifies the VRF instance to associate with the BGP peer. MetalLB can advertise services and make routing decisions based on the routing information in the VRF.

b. Apply the configuration for the BGP peer by running the following command:

```
$ oc apply -f frr-via-vrf.yaml
```

3. Create an **IPAddressPool** CR:

a. Create a file, such as **first-pool.yaml**, with content like the following example:

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: first-pool
  namespace: metallb-system
spec:
  addresses:
  - 192.169.10.0/32
# ...
```

b. Apply the configuration for the IP address pool by running the following command:

```
$ oc apply -f first-pool.yaml
```

4. Create a **BGPAdvertisement** CR:

a. Create a file, such as **first-adv.yaml**, with content like the following example:

```
apiVersion: metallb.io/v1beta1
kind: BGPAdvertisement
```

```
  metadata:
    name: first-adv
    namespace: metallb-system
  spec:
    ipAddressPools:
      - first-pool
    peers:
      - frrviavrf
    nodeSelectors:
      - matchLabels:
          egress-service.k8s.ovn.org/test-server1: ""
  # ...
```

where:

peers

In this example, MetalLB advertises a range of IP addresses from the **first-pool** IP address pool to the **frrviavrf** BGP peer.

nodeSelectors

In this example, the **EgressService** CR configures the source IP address for egress traffic to use the load-balancer service IP address. Therefore, you must specify the load-balancer node for return traffic to use the same return path for the traffic originating from the pod.

b. Apply the configuration for the BGP advertisement by running the following command:

```
$ oc apply -f first-adv.yaml
```

5. Create an **EgressService** CR:

a. Create a file, such as **egress-service.yaml**, with content like the following example:

```
apiVersion: k8s.ovn.org/v1
kind: EgressService
metadata:
  name: server1
  namespace: test
spec:
  sourceIPBy: "LoadBalancerIP"
  nodeSelector:
    matchLabels:
      vrf: "true"
  network: "2"
# ...
```

where:

metadata.name

Specifies the name for the egress service. The name of the **EgressService** resource must match the name of the load-balancer service that you want to modify.

metadata.namespace

Specifies the namespace for the egress service. The namespace for the **EgressService** must match the namespace of the load-balancer service that you want to modify. The egress service is namespace-scoped.

spec.sourceIPBy

Specifies the **LoadBalancer** service ingress IP address as the source IP address for egress traffic.

matchLabels.vrf

If you specify **LoadBalancer** for the **sourceIPBy** specification, a single node handles the **LoadBalancer** service traffic. In this example, only a node with the label **vrf: "true"** can handle the service traffic. If you do not specify a node, OVN-Kubernetes selects a worker node to handle the service traffic. When a node is selected, OVN-Kubernetes labels the node in the following format: **egress-service.k8s.ovn.org/<svc_namespace>-<svc_name>: ""**.

network

Specifyies the routing table ID for egress traffic. Ensure that the value matches the **route-table-id** ID defined in the **NodeNetworkConfigurationPolicy** resource, for example, **route-table-id: 2**.

b. Apply the configuration for the egress service by running the following command:

```
$ oc apply -f egress-service.yaml
```

Verification

1. Verify that you can access the application endpoint of the pods running behind the MetalLB service by running the following command:

```
$ curl <external_ip_address>:<port_number>
```

- **<external_ip_address>:<port_number>**: Specifies the external IP address and port number to suit your application endpoint.

2. Optional: If you assigned the **LoadBalancer** service ingress IP address as the source IP address for egress traffic, verify this configuration by using tools such as **tcpdump** to analyze packets received at the external client.

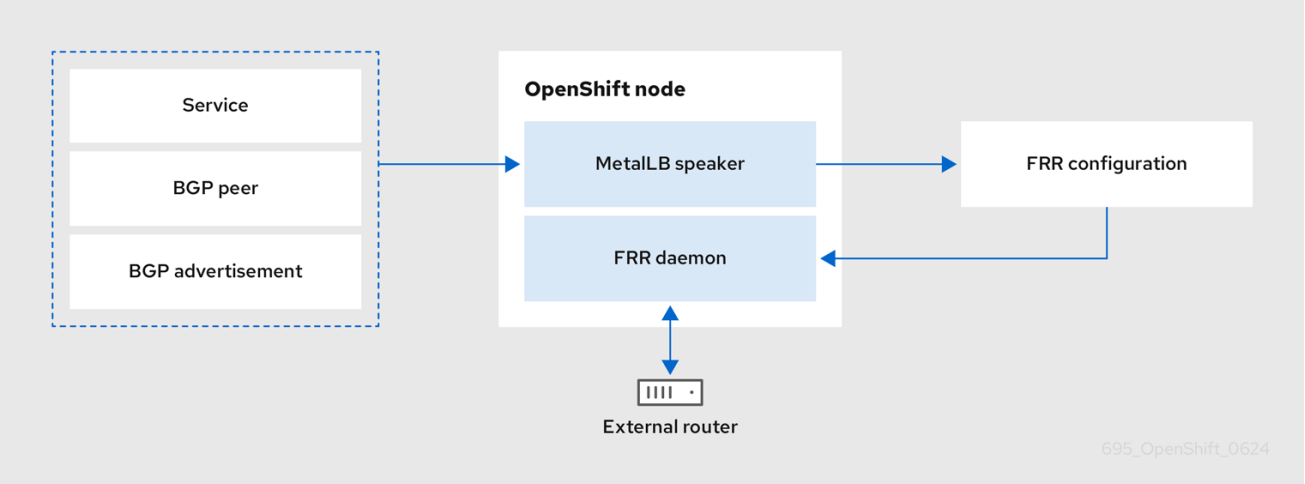
Additional resources

- About virtual routing and forwarding
- Exposing a service through a network VRF
- Example: Network interface with a VRF instance node network configuration policy
- Configuring an egress service

4.8. CONFIGURING THE INTEGRATION OF METALLB AND FRR-K8S

To access advanced routing services not natively provided by MetalLB, configure the **FRRConfiguration** custom resource (CR). Defining the CR exposes specific FRRouting (FRR) capabilities and extends the routing functionality of your cluster beyond standard MetalLB advertisements.

FRRouting (FRR) is a free, open-source internet routing protocol suite for Linux and UNIX platforms. **FRR-K8s** is a Kubernetes-based DaemonSet that exposes a subset of the **FRR** API in a Kubernetes-compliant manner. **MetalLB** generates the **FRR-K8s** configuration corresponding to the MetalLB configuration applied.



WARNING

When configuring Virtual Route Forwarding (VRF), you must change the VRFs to a table ID lower than **1000** as higher than **1000** is reserved for OpenShift Container Platform.

4.8.1. FRR configurations

You can create multiple **FRRConfiguration** CRs to use **FRR** services in **MetalLB**.

MetalLB generates an **FRRConfiguration** object which **FRR-K8s** merges with all other configurations that all users have created. For example, you can configure **FRR-K8s** to receive all of the prefixes advertised by a given neighbor. The following example configures **FRR-K8s** to receive all of the prefixes advertised by a **BGPPeer** with host **172.18.0.5**:

Example FRRConfiguration CR

```
apiVersion: frrk8s.metallb.io/v1beta1
kind: FRRConfiguration
metadata:
 name: test
 namespace: metallb-system
spec:
 bgp:
   routers:
   - asn: 64512
     neighbors:
     - address: 172.18.0.5
       asn: 64512
```

```
      toReceive:
       allowed:
          mode: all
# ...
```

You can also configure FRR-K8s to always block a set of prefixes, regardless of the configuration applied. This can be useful to avoid routes going to pods or **ClusterIPs** CIDRs that might result in cluster malfunctions. The following example blocks the set of prefixes **192.168.1.0/24**:

Example MetalLB CR

```
apiVersion: metallb.io/v1beta1
kind: MetalLB
metadata:
  name: metallb
  namespace: metallb-system
spec:
  bgpBackend: frr-k8s
  frrk8sConfig:
    alwaysBlock:
    - 192.168.1.0/24
# ...
```

You can set **FRR-K8s** to block the **clusterNetwork** CIDR and **serviceNetwork** CIDR. You can view the values for these CIDR address specifications by running the following command:

```
$ oc describe network.config/cluster
```

4.8.2. Configuring the FRRConfiguration CRD

To customize routing behavior beyond standard MetalLB capabilities, configure the **FRRConfiguration** custom resource (CR).

The following reference examples demonstrate how to define specific FRRouting (FRR) parameters to enable advanced services, such as receiving routes:

The **routers** parameter

You can use the **routers** parameter to configure multiple routers, one for each Virtual Routing and Forwarding (VRF) resource. For each router, you must define the Autonomous System Number (ASN).

You can also define a list of Border Gateway Protocol (BGP) neighbors to connect to, as in the following example:

Example FRRConfiguration CR

```
apiVersion: frrk8s.metallb.io/v1beta1
kind: FRRConfiguration
metadata:
  name: test
  namespace: frr-k8s-system
spec:
  bgp:
    routers:
```

```
    - asn: 64512
      neighbors:
      - address: 172.30.0.3
        asn: 4200000000
        ebgpMultiHop: true
        port: 180
      - address: 172.18.0.6
        asn: 4200000000
        port: 179
# ...
```

The **toAdvertise** parameter

By default, **FRR-K8s** does not advertise the prefixes configured as part of a router configuration. To advertise the prefixes, you use the **toAdvertise** parameter.

You can advertise a subset of the prefixes, as in the following example:

Example FRRConfiguration CR

```
apiVersion: frrk8s.metallb.io/v1beta1
kind: FRRConfiguration
metadata:
  name: test
  namespace: frr-k8s-system
spec:
  bgp:
    routers:
    - asn: 64512
      neighbors:
      - address: 172.30.0.3
        asn: 4200000000
        ebgpMultiHop: true
        port: 180
        toAdvertise:
          allowed:
            prefixes:
            - 192.168.2.0/24
      prefixes:
        - 192.168.2.0/24
        - 192.169.2.0/24
# ...
```

- **allowed.prefixes**: Advertises a subset of prefixes.

The following example shows you how to advertise all of the prefixes:

Example FRRConfiguration CR

```
apiVersion: frrk8s.metallb.io/v1beta1
kind: FRRConfiguration
metadata:
  name: test
  namespace: frr-k8s-system
spec:
  bgp:
```

```
    routers:
    - asn: 64512
      neighbors:
      - address: 172.30.0.3
        asn: 4200000000
        ebgpMultiHop: true
        port: 180
        toAdvertise:
          allowed:
            mode: all
      prefixes:
        - 192.168.2.0/24
        - 192.169.2.0/24
# ...
```

- **allowed.mode**: Advertises all prefixes.

  The toReceive parameter

  By default, **FRR-K8s** does not process any prefixes advertised by a neighbor. You can use the **toReceive** parameter to process such addresses.

You can configure for a subset of the prefixes, as in this example:

Example FRRConfiguration CR

```
apiVersion: frrk8s.metallb.io/v1beta1
kind: FRRConfiguration
metadata:
  name: test
  namespace: frr-k8s-system
spec:
  bgp:
    routers:
    - asn: 64512
      neighbors:
      - address: 172.18.0.5
          asn: 64512
          port: 179
          toReceive:
            allowed:
              prefixes:
              - prefix: 192.168.1.0/24
              - prefix: 192.169.2.0/24
                ge: 25
                le: 28
# ...
```

- **prefixes**: The prefix is applied if the prefix length is less than or equal to the **le** prefix length and greater than or equal to the **ge** prefix length.

The following example configures FRR to handle all the prefixes announced:

Example FRRConfiguration CR

```
apiVersion: frrk8s.metallb.io/v1beta1
```

```
kind: FRRConfiguration
metadata:
  name: test
  namespace: frr-k8s-system
spec:
  bgp:
    routers:
    - asn: 64512
      neighbors:
      - address: 172.18.0.5
          asn: 64512
          port: 179
          toReceive:
            allowed:
              mode: all
# ...
```

The **bgp** parameter

You can use the **bgp** parameter to define various **BFD** profiles and associate them with a neighbor. In the following example, **BFD** backs up the **BGP** session and **FRR** can detect link failures:

Example FRRConfiguration CR

```
apiVersion: frrk8s.metallb.io/v1beta1
kind: FRRConfiguration
metadata:
  name: test
  namespace: frr-k8s-system
spec:
  bgp:
    routers:
    - asn: 64512
      neighbors:
      - address: 172.30.0.3
        asn: 64512
        port: 180
        bfdProfile: defaultprofile
    bfdProfiles:
      - name: defaultprofile
# ...
```

The **nodeSelector** parameter

By default, **FRR-K8s** applies the configuration to all nodes where the daemon is running. You can use the **nodeSelector** parameter to specify the nodes to which you want to apply the configuration. For example:

Example FRRConfiguration CR

```
apiVersion: frrk8s.metallb.io/v1beta1
kind: FRRConfiguration
metadata:
  name: test
  namespace: frr-k8s-system
spec:
```

```
  bgp:
    routers:
    - asn: 64512
  nodeSelector:
    labelSelector:
    foo: "bar"
# ...
```

The fields for the **FRRConfiguration** custom resource are described in the following table:

Table 4.9. MetalLB FRRConfiguration custom resource

Parameter	Type	Description
spec.bgp.routers	**array**	Specifies the routers that FRR is to configure (one per VRF).
spec.bgp.routers.asn	**integer**	The Autonomous System Number (ASN) to use for the local end of the session.
spec.bgp.routers.id	**string**	Specifies the ID of the **bgp** router.
spec.bgp.routers.vrf	**string**	Specifies the host VRF used to establish sessions from this router.
spec.bgp.routers.neighbors	**array**	Specifies the neighbors to establish BGP sessions with.
spec.bgp.routers.neighbors.asn	**integer**	Specifies the ASN to use for the remote end of the session. If you use this parameter, you cannot specify a value in the **spec.bgp.routers.neighbors.dynamicASN** parameter.
spec.bgp.routers.neighbors.dynamicASN	**string**	Detects the ASN to use for the remote end of the session without explicitly setting it. Specify **internal** for a neighbor with the same ASN, or **external** for a neighbor with a different ASN. If you use this parameter, you cannot specify a value in the **spec.bgp.routers.neighbors.asn** parameter.
spec.bgp.routers.neighbors.address	**string**	Specifies the IP address to establish the session with.
spec.bgp.routers.neighbors.port	**integer**	Specifies the port to dial when establishing the session. Defaults to **179**.
spec.bgp.routers.neighbors.password	**string**	Specifies the password to use for establishing the BGP session. **Password** and **PasswordSecret** are mutually exclusive.

Parameter	Type	Description
spec.bgp.routers.neighbors.passwordSecret	**string**	Specifies the name of the authentication secret for the neighbor. The secret must be of type "kubernetes.io/basic-auth", and in the same namespace as the FRR-K8s daemon. The key "password" stores the password in the secret. **Password** and **PasswordSecret** are mutually exclusive.
spec.bgp.routers.neighbors.holdTime	**duration**	Specifies the requested BGP hold time, per RFC4271. Defaults to 180s.
spec.bgp.routers.neighbors.keepaliveTime	**duration**	Specifies the requested BGP keepalive time, per RFC4271. Defaults to **60s**.
spec.bgp.routers.neighbors.connectTime	**duration**	Specifies how long BGP waits between connection attempts to a neighbor.
spec.bgp.routers.neighbors.ebgpMultiHop	**boolean**	Indicates if the BGPPeer is a multi-hop away.
spec.bgp.routers.neighbors.bfdProfile	**string**	Specifies the name of the BFD Profile to use for the BFD session associated with the BGP session. If not set, the BFD session is not set up.
spec.bgp.routers.neighbors.toAdvertise.allowed	**array**	Represents the list of prefixes to advertise to a neighbor, and the associated properties.
spec.bgp.routers.neighbors.toAdvertise.allowed.prefixes	**string array**	Specifies the list of prefixes to advertise to a neighbor. This list must match the prefixes that you define in the router.
spec.bgp.routers.neighbors.toAdvertise.allowed.mode	**string**	Specifies the mode to use when handling the prefixes. You can set to **filtered** to allow only the prefixes in the prefixes list. You can set to **all** to allow all the prefixes configured on the router.
spec.bgp.routers.neighbors.toAdvertise.withLocalPref	**array**	Specifies the prefixes associated with an advertised local preference. You must specify the prefixes associated with a local preference in the prefixes allowed to be advertised.

Parameter	Type	Description
spec.bgp.routers.neighbors.toAdvertise.withLocalPref.prefixes	**string array**	Specifies the prefixes associated with the local preference.
spec.bgp.routers.neighbors.toAdvertise.withLocalPref.localPref	**integer**	Specifies the local preference associated with the prefixes.
spec.bgp.routers.neighbors.toAdvertise.withCommunity	**array**	Specifies the prefixes associated with an advertised BGP community. You must include the prefixes associated with a local preference in the list of prefixes that you want to advertise.
spec.bgp.routers.neighbors.toAdvertise.withCommunity.prefixes	**string array**	Specifies the prefixes associated with the community.
spec.bgp.routers.neighbors.toAdvertise.withCommunity.community	**string**	Specifies the community associated with the prefixes.
spec.bgp.routers.neighbors.toReceive	**array**	Specifies the prefixes to receive from a neighbor.
spec.bgp.routers.neighbors.toReceive.allowed	**array**	Specifies the information that you want to receive from a neighbor.
spec.bgp.routers.neighbors.toReceive.allowed.prefixes	**array**	Specifies the prefixes allowed from a neighbor.
spec.bgp.routers.neighbors.toReceive.allowed.mode	**string**	Specifies the mode to use when handling the prefixes. When set to **filtered**, only the prefixes in the **prefixes** list are allowed. When set to **all**, all the prefixes configured on the router are allowed.

Parameter	Type	Description
spec.bgp.routers.neighbors.disableMP	**boolean**	Disables MP BGP to prevent it from separating IPv4 and IPv6 route exchanges into distinct BGP sessions.
spec.bgp.routers.prefixes	**string array**	Specifies all prefixes to advertise from this router instance.
spec.bgp.bfdProfiles	**array**	Specifies the list of BFD profiles to use when configuring the neighbors.
spec.bgp.bfdProfiles.name	**string**	The name of the BFD Profile to be referenced in other parts of the configuration.
spec.bgp.bfdProfiles.receiveInterval	**integer**	Specifies the minimum interval at which this system can receive control packets, in milliseconds. Defaults to **300ms**.
spec.bgp.bfdProfiles.transmitInterval	**integer**	Specifies the minimum transmission interval, excluding jitter, that this system wants to use to send BFD control packets, in milliseconds. Defaults to **300ms**.
spec.bgp.bfdProfiles.detectMultiplier	**integer**	Configures the detection multiplier to determine packet loss. To determine the connection loss-detection timer, multiply the remote transmission interval by this value.
spec.bgp.bfdProfiles.echoInterval	**integer**	Configures the minimal echo receive transmission-interval that this system can handle, in milliseconds. Defaults to **50ms**.
spec.bgp.bfdProfiles.echoMode	**boolean**	Enables or disables the echo transmission mode. This mode is disabled by default, and not supported on multihop setups.
spec.bgp.bfdProfiles.passiveMode	**boolean**	Mark session as passive. A passive session does not attempt to start the connection and waits for control packets from peers before it begins replying.
spec.bgp.bfdProfiles.MinimumTtl	**integer**	For multihop sessions only. Configures the minimum expected TTL for an incoming BFD control packet.
spec.nodeSelector	**string**	Limits the nodes that attempt to apply this configuration. If specified, only those nodes whose labels match the specified selectors attempt to apply the configuration. If it is not specified, all nodes attempt to apply this configuration.

Parameter	Type	Description
status	**string**	Defines the observed state of FRRConfiguration.

4.8.3. How FRR-K8s merges multiple configurations

FRR-K8s uses an additive merge strategy when multiple users configure the same node. By using FRR-K8s, you can extend existing configurations, such as adding neighbors or prefixes, but prevent the removal of components defined by other sources.

Configuration conflicts

Certain configurations can cause conflicts, leading to errors, for example:

- different ASN for the same router (in the same VRF)
- different ASN for the same neighbor (with the same IP / port)
- multiple BFD profiles with the same name but different values

When the daemon finds an invalid configuration for a node, it reports the configuration as invalid and reverts to the previous valid **FRR** configuration.

Merging

When merging, you can complete the following actions:

- Extend the set of IP addresses that you want to advertise to a neighbor.
- Add an extra neighbor with its set of IP addresses.
- Extend the set of IP addresses to which you want to associate a community.
- Allow incoming routes for a neighbor.

Each configuration must be self contained. This means, for example, that you cannot allow prefixes that are not defined in the router section by leveraging prefixes coming from another configuration.

If the configurations to be applied are compatible, merging works as follows:

- **FRR-K8s** combines all the routers.
- **FRR-K8s** merges all prefixes and neighbors for each router.
- **FRR-K8s** merges all filters for each neighbor.

NOTE

A less restrictive filter has precedence over a stricter one. For example, a filter accepting some prefixes has precedence over a filter not accepting any, and a filter accepting all prefixes has precedence over one that accepts some.

4.9. METALLB LOGGING, TROUBLESHOOTING, AND SUPPORT

To diagnose and resolve MetalLB configuration issues, refer to this list of commonly used commands. By using these commands, you can verify network connectivity and inspect service states to ensure efficient error recovery.

4.9.1. Setting the MetalLB logging levels

To manage log verbosity for the FRRouting (FRR) container, configure the **logLevel** specification. By adjusting this setting, you can reduce log volume from the default info level or increase detail for troubleshooting MetalLB configuration issues.

Gain a deeper insight into MetalLB by setting the **logLevel** to **debug**.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- You have installed the OpenShift CLI (**oc**).

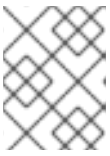
Procedure

1. Create a file, such as **setdebugloglevel.yaml**, with content like the following example:

```
apiVersion: metallb.io/v1beta1
kind: MetalLB
metadata:
  name: metallb
  namespace: metallb-system
spec:
  logLevel: debug
  nodeSelector:
    node-role.kubernetes.io/worker: ""
```

2. Apply the configuration by entering the following command:

```
$ oc replace -f setdebugloglevel.yaml
```



NOTE

Use the **oc replace** command because the **metallb** CR was already created and you need to change only the log level.

3. Display the names of the **speaker** pods:

```
$ oc get -n metallb-system pods -l component=speaker
```

Example output

```
NAME                READY   STATUS    RESTARTS   AGE
speaker-2m9pm         4/4     Running   0          9m19s
speaker-7m4qw         3/4     Running   0          19s
speaker-szlmx         4/4     Running   0          9m19s
```

NOTE

Speaker and controller pods are recreated to ensure the updated logging level is applied. The logging level is modified for all the components of MetalLB.

4. View the **speaker** logs:

```
$ oc logs -n metallb-system speaker-7m4qw -c speaker
```

Example output

```
{"branch":"main","caller":"main.go:92","commit":"3d052535","goversion":"gc / go1.17.1 /
amd64","level":"info","msg":"MetalLB speaker starting (commit 3d052535, branch
main)","ts":"2022-05-17T09:55:05Z","version":""}
{"caller":"announcer.go:110","event":"createARPResponder","interface":"ens4","level":"info","m
sg":"created ARP responder for interface","ts":"2022-05-17T09:55:05Z"}
{"caller":"announcer.go:119","event":"createNDPResponder","interface":"ens4","level":"info","m
sg":"created NDP responder for interface","ts":"2022-05-17T09:55:05Z"}
{"caller":"announcer.go:110","event":"createARPResponder","interface":"tun0","level":"info","ms
g":"created ARP responder for interface","ts":"2022-05-17T09:55:05Z"}
{"caller":"announcer.go:119","event":"createNDPResponder","interface":"tun0","level":"info","m
sg":"created NDP responder for interface","ts":"2022-05-17T09:55:05Z"}
I0517 09:55:06.515686      95 request.go:665] Waited for 1.026500832s due to client-side
throttling, not priority and fairness, request:
GET:https://172.30.0.1:443/apis/operators.coreos.com/v1alpha1?timeout=32s
{"Starting Manager":"(MISSING)","caller":"k8s.go:389","level":"info","ts":"2022-05-
17T09:55:08Z"}
{"caller":"speakerlist.go:310","level":"info","msg":"node event - forcing sync","node
addr":"10.0.128.4","node event":"NodeJoin","node name":"ci-ln-qb8t3mb-72292-7s7rh-
worker-a-vvznj","ts":"2022-05-17T09:55:08Z"}
{"caller":"service_controller.go:113","controller":"ServiceReconciler","enqueueing":"openshift-
kube-controller-manager-operator/metrics","epslice":"{\"metadata\":{\"name\":\"metrics-
xtsxr\",\"generateName\":\"metrics-\",\"namespace\":\"openshift-kube-controller-manager-
operator\",\"uid\":\"ac6766d7-8504-492c-9d1e-
4ae8897990ad\",\"resourceVersion\":\"9041\",\"generation\":4,\"creationTimestamp\":\"2022-
05-17T07:16:53Z\",\"labels\":{\"app\":\"kube-controller-manager-
operator\",\"endpointslice.kubernetes.io/managed-by\":\"endpointslice-
controller.k8s.io\",\"kubernetes.io/service-name\":\"metrics\"},\"annotations\":
{\"endpoints.kubernetes.io/last-change-trigger-time\":\"2022-05-
17T07:21:34Z\"},\"ownerReferences\":
[{\"apiVersion\":\"v1\",\"kind\":\"Service\",\"name\":\"metrics\",\"uid\":\"0518eed3-6152-42be-
b566-0bd00a60faf8\",\"controller\":true,\"blockOwnerDeletion\":true}],\"managedFields\":
[{\"manager\":\"kube-controller-
manager\",\"operation\":\"Update\",\"apiVersion\":\"discovery.k8s.io/v1\",\"time\":\"2022-05-
17T07:20:02Z\",\"fieldsType\":\"FieldsV1\",\"fieldsV1\":{\"f:addressType\":{},\"f:endpoints\":
{},\"f:metadata\":{\"f:annotations\":{\".\":{},\"f:endpoints.kubernetes.io/last-change-trigger-
time\":{}},\"f:generateName\":{},\"f:labels\":{\".\":{},\"f:app\":
{},\"f:endpointslice.kubernetes.io/managed-by\":{},\"f:kubernetes.io/service-name\":
{}},\"f:ownerReferences\":{\".\":{},\"k:{\\\"uid\\\":\\\"0518eed3-6152-42be-b566-
0bd00a60faf8\\\"}\":{}}},\"f:ports\":{}}}]},\"addressType\":\"IPv4\",\"endpoints\":[{\"addresses\":
[\"10.129.0.7\"],\"conditions\":{\"ready\":true,\"serving\":true,\"terminating\":false},\"targetRef\":
{\"kind\":\"Pod\",\"namespace\":\"openshift-kube-controller-manager-
operator\",\"name\":\"kube-controller-manager-operator-6b98b89ddd-
8d4nf\",\"uid\":\"dd5139b8-e41c-4946-a31b-
1a629314e844\",\"resourceVersion\":\"9038\"},\"nodeName\":\"ci-ln-qb8t3mb-72292-7s7rh-
```

```
master-0\",\"zone\":\"us-central1-a\"}],\"ports\":
[{\"name\":\"https\",\"protocol\":\"TCP\",\"port\":8443}]}","level":"debug","ts":"2022-05-
17T09:55:08Z"}
```

5. View the FRR logs:

```
$ oc logs -n metallb-system speaker-7m4qw -c frr
```

Example output

```
Started watchfrr
2022/05/17 09:55:05 ZEBRA: client 16 says hello and bids fair to announce only bgp routes
vrf=0
2022/05/17 09:55:05 ZEBRA: client 31 says hello and bids fair to announce only vnc routes
vrf=0
2022/05/17 09:55:05 ZEBRA: client 38 says hello and bids fair to announce only static routes
vrf=0
2022/05/17 09:55:05 ZEBRA: client 43 says hello and bids fair to announce only bfd routes
vrf=0
2022/05/17 09:57:25.089 BGP: Creating Default VRF, AS 64500
2022/05/17 09:57:25.090 BGP: dup addr detect enable max_moves 5 time 180 freeze
disable freeze_time 0
2022/05/17 09:57:25.090 BGP: bgp_get: Registering BGP instance (null) to zebra
2022/05/17 09:57:25.090 BGP: Registering VRF 0
2022/05/17 09:57:25.091 BGP: Rx Router Id update VRF 0 Id 10.131.0.1/32
2022/05/17 09:57:25.091 BGP: RID change : vrf VRF default(0), RTR ID 10.131.0.1
2022/05/17 09:57:25.091 BGP: Rx Intf add VRF 0 IF br0
2022/05/17 09:57:25.091 BGP: Rx Intf add VRF 0 IF ens4
2022/05/17 09:57:25.091 BGP: Rx Intf address add VRF 0 IF ens4 addr 10.0.128.4/32
2022/05/17 09:57:25.091 BGP: Rx Intf address add VRF 0 IF ens4 addr
fe80::c9d:84da:4d86:5618/64
2022/05/17 09:57:25.091 BGP: Rx Intf add VRF 0 IF lo
2022/05/17 09:57:25.091 BGP: Rx Intf add VRF 0 IF ovs-system
2022/05/17 09:57:25.091 BGP: Rx Intf add VRF 0 IF tun0
2022/05/17 09:57:25.091 BGP: Rx Intf address add VRF 0 IF tun0 addr 10.131.0.1/23
2022/05/17 09:57:25.091 BGP: Rx Intf address add VRF 0 IF tun0 addr
fe80::40f1:d1ff:feb6:5322/64
2022/05/17 09:57:25.091 BGP: Rx Intf add VRF 0 IF veth2da49fed
2022/05/17 09:57:25.091 BGP: Rx Intf address add VRF 0 IF veth2da49fed addr
fe80::24bd:d1ff:fec1:d88/64
2022/05/17 09:57:25.091 BGP: Rx Intf add VRF 0 IF veth2fa08c8c
2022/05/17 09:57:25.091 BGP: Rx Intf address add VRF 0 IF veth2fa08c8c addr
fe80::6870:ff:fe96:efc8/64
2022/05/17 09:57:25.091 BGP: Rx Intf add VRF 0 IF veth41e356b7
2022/05/17 09:57:25.091 BGP: Rx Intf address add VRF 0 IF veth41e356b7 addr
fe80::48ff:37ff:fede:eb4b/64
2022/05/17 09:57:25.092 BGP: Rx Intf add VRF 0 IF veth1295c6e2
2022/05/17 09:57:25.092 BGP: Rx Intf address add VRF 0 IF veth1295c6e2 addr
fe80::b827:a2ff:feed:637/64
2022/05/17 09:57:25.092 BGP: Rx Intf add VRF 0 IF veth9733c6dc
2022/05/17 09:57:25.092 BGP: Rx Intf address add VRF 0 IF veth9733c6dc addr
fe80::3cf4:15ff:fe11:e541/64
2022/05/17 09:57:25.092 BGP: Rx Intf add VRF 0 IF veth336680ea
2022/05/17 09:57:25.092 BGP: Rx Intf address add VRF 0 IF veth336680ea addr
fe80::94b1:8bff:fe7e:488c/64
```

```
2022/05/17 09:57:25.092 BGP: Rx Intf add VRF 0 IF vetha0a907b7
2022/05/17 09:57:25.092 BGP: Rx Intf address add VRF 0 IF vetha0a907b7 addr
fe80::3855:a6ff:fe73:46c3/64
2022/05/17 09:57:25.092 BGP: Rx Intf add VRF 0 IF vethf35a4398
2022/05/17 09:57:25.092 BGP: Rx Intf address add VRF 0 IF vethf35a4398 addr
fe80::40ef:2fff:fe57:4c4d/64
2022/05/17 09:57:25.092 BGP: Rx Intf add VRF 0 IF vethf831b7f4
2022/05/17 09:57:25.092 BGP: Rx Intf address add VRF 0 IF vethf831b7f4 addr
fe80::f0d9:89ff:fe7c:1d32/64
2022/05/17 09:57:25.092 BGP: Rx Intf add VRF 0 IF vxlan_sys_4789
2022/05/17 09:57:25.092 BGP: Rx Intf address add VRF 0 IF vxlan_sys_4789 addr
fe80::80c1:82ff:fe4b:f078/64
2022/05/17 09:57:26.094 BGP: 10.0.0.1 [FSM] Timer (start timer expire).
2022/05/17 09:57:26.094 BGP: 10.0.0.1 [FSM] BGP_Start (Idle->Connect), fd -1
2022/05/17 09:57:26.094 BGP: Allocated bnc 10.0.0.1/32(0)(VRF default) peer
0x7f807f7631a0
2022/05/17 09:57:26.094 BGP: sendmsg_zebra_rnh: sending cmd
ZEBRA_NEXTHOP_REGISTER for 10.0.0.1/32 (vrf VRF default)
2022/05/17 09:57:26.094 BGP: 10.0.0.1 [FSM] Waiting for NHT
2022/05/17 09:57:26.094 BGP: bgp_fsm_change_status : vrf default(0), Status: Connect
established_peers 0
2022/05/17 09:57:26.094 BGP: 10.0.0.1 went from Idle to Connect
2022/05/17 09:57:26.094 BGP: 10.0.0.1 [FSM] TCP_connection_open_failed (Connect-
>Active), fd -1
2022/05/17 09:57:26.094 BGP: bgp_fsm_change_status : vrf default(0), Status: Active
established_peers 0
2022/05/17 09:57:26.094 BGP: 10.0.0.1 went from Connect to Active
2022/05/17 09:57:26.094 ZEBRA: rnh_register msg from client bgp: hdr->length=8,
type=nexthop vrf=0
2022/05/17 09:57:26.094 ZEBRA: 0: Add RNH 10.0.0.1/32 type Nexthop
2022/05/17 09:57:26.094 ZEBRA: 0:10.0.0.1/32: Evaluate RNH, type Nexthop (force)
2022/05/17 09:57:26.094 ZEBRA: 0:10.0.0.1/32: NH has become unresolved
2022/05/17 09:57:26.094 ZEBRA: 0: Client bgp registers for RNH 10.0.0.1/32 type Nexthop
2022/05/17 09:57:26.094 BGP: VRF default(0): Rcvd NH update 10.0.0.1/32(0) - metric 0/0
#nhops 0/0 flags 0x6
2022/05/17 09:57:26.094 BGP: NH update for 10.0.0.1/32(0)(VRF default) - flags 0x6
chgflags 0x0 - evaluate paths
2022/05/17 09:57:26.094 BGP: evaluate_paths: Updating peer (10.0.0.1(VRF default)) status
with NHT
2022/05/17 09:57:30.081 ZEBRA: Event driven route-map update triggered
2022/05/17 09:57:30.081 ZEBRA: Event handler for route-map: 10.0.0.1-out
2022/05/17 09:57:30.081 ZEBRA: Event handler for route-map: 10.0.0.1-in
2022/05/17 09:57:31.104 ZEBRA: netlink_parse_info: netlink-listen (NS 0) type
RTM_NEWNEIGH(28), len=76, seq=0, pid=0
2022/05/17 09:57:31.104 ZEBRA:  Neighbor Entry received is not on a VLAN or a BRIDGE,
ignoring
2022/05/17 09:57:31.105 ZEBRA: netlink_parse_info: netlink-listen (NS 0) type
RTM_NEWNEIGH(28), len=76, seq=0, pid=0
2022/05/17 09:57:31.105 ZEBRA:  Neighbor Entry received is not on a VLAN or a BRIDGE,
ignoring
```

4.9.1.1. FRRouting (FRR) log levels

To control the verbosity of network logs for troubleshooting or monitoring, refer to the FRRouting (FRR) logging levels.

The following values define the severity of recorded events, so that you can use them to filter output based on operational requirements:

Table 4.10. Log levels

Log level	Description
all	Supplies all logging information for all logging levels.
debug	Information that is diagnostically helpful to people. Set to **debug** to give detailed troubleshooting information.
info	Provides information that always should be logged but under normal circumstances does not require user intervention. This is the default logging level.
warn	Anything that can potentially cause inconsistent **MetalLB** behaviour. Usually **MetalLB** automatically recovers from this type of error.
error	Any error that is fatal to the functioning of **MetalLB**. These errors usually require administrator intervention to fix.
none	Turn off all logging.

4.9.2. Troubleshooting BGP issues

To diagnose and resolve BGP configuration issues, execute commands directly within the FRR container. By accessing the container, you can verify routing states and identify connectivity errors.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Display the names of the **frr-k8s** pods by running the following command:

```
$ oc -n metallb-system get pods -l component=frr-k8s
```

Example output

```
NAME          READY   STATUS    RESTARTS   AGE
frr-k8s-thsmw   6/6     Running   0          109m
```

2. Display the running configuration for FRR by running the following command:

```
$ oc exec -n metallb-system frr-k8s-thsmw -c frr -- vtysh -c "show running-config"
```

Example output

```
Building configuration...

Current configuration:
!
frr version 8.5.3
frr defaults traditional
hostname some-hostname
log file /etc/frr/frr.log informational
log timestamp precision 3
no ip forwarding
no ipv6 forwarding
service integrated-vtysh-config
!
router bgp 64500
 bgp router-id 10.0.1.2
 no bgp ebgp-requires-policy
 no bgp default ipv4-unicast
 no bgp network import-check
 neighbor 10.0.2.3 remote-as 64500
 neighbor 10.0.2.3 bfd profile doc-example-bfd-profile-full
 neighbor 10.0.2.3 timers 5 15
 neighbor 10.0.2.4 remote-as 64500
 neighbor 10.0.2.4 bfd profile doc-example-bfd-profile-full
 neighbor 10.0.2.4 timers 5 15
 !
 address-family ipv4 unicast
  network 203.0.113.200/30
  neighbor 10.0.2.3 activate
  neighbor 10.0.2.3 route-map 10.0.2.3-in in
  neighbor 10.0.2.4 activate
  neighbor 10.0.2.4 route-map 10.0.2.4-in in
 exit-address-family
 !
 address-family ipv6 unicast
  network fc00:f853:ccd:e799::/124
  neighbor 10.0.2.3 activate
  neighbor 10.0.2.3 route-map 10.0.2.3-in in
  neighbor 10.0.2.4 activate
  neighbor 10.0.2.4 route-map 10.0.2.4-in in
 exit-address-family
!
route-map 10.0.2.3-in deny 20
!
route-map 10.0.2.4-in deny 20
!
ip nht resolve-via-default
!
ipv6 nht resolve-via-default
!
line vty
!
bfd
 profile doc-example-bfd-profile-full
  transmit-interval 35
```

```
  receive-interval 35
  passive-mode
  echo-mode
  echo-interval 35
  minimum-ttl 10
 !
!
end
```

where:

router bgp 64500

Specifies the **router bgp** that indicates the ASN for MetalLB.

neighbor 10.0.2.3 remote-as 64500

Specifies that a **neighbor <ip-address> remote-as <peer-ASN>** line exists for each BGP peer custom resource that you added.

bfd profile doc-example-bfd-profile-full

Specifies that the BFD profile is associated with the correct BGP peer and that the BFD profile shows in the command output.

network 203.0.113.200/30

Specifies that the **network <ip-address-range>** lines match the IP address ranges that you specified in address pool custom resources

3. Display the BGP summary by running the following command:

```
$ oc exec -n metallb-system frr-k8s-thsmw -c frr -- vtysh -c "show bgp summary"
```

Example output

```
IPv4 Unicast Summary:
BGP router identifier 10.0.1.2, local AS number 64500 vrf-id 0
BGP table version 1
RIB entries 1, using 192 bytes of memory
Peers 2, using 29 KiB of memory

Neighbor        V         AS   MsgRcvd   MsgSent   TblVer  InQ OutQ  Up/Down State/PfxRcd
PfxSnt
10.0.2.3        4      64500       387       389        0    0    0 00:32:02            0        1
10.0.2.4        4      64500         0         0        0    0    0    never       Active        0

Total number of neighbors 2

IPv6 Unicast Summary:
BGP router identifier 10.0.1.2, local AS number 64500 vrf-id 0
BGP table version 1
RIB entries 1, using 192 bytes of memory
Peers 2, using 29 KiB of memory

Neighbor        V         AS   MsgRcvd   MsgSent   TblVer  InQ OutQ  Up/Down State/PfxRcd
PfxSnt
10.0.2.3        4      64500       387       389        0    0    0 00:32:02 NoNeg
```

```
10.0.2.4       4     64500       0       0       0    0    0    never       Active       0

Total number of neighbors 2
```

where:

10.0.2.3

Specifies that the output includes a line for each BGP peer custom resource that you added.

10.0.2.4

Specifies that the output shows **0** messages received and **0** messages sent, which indicates a BGP peer that does not have a BGP session. Check network connectivity and the BGP configuration of the BGP peer.

4. Display the BGP peers that received an address pool by running the following command:

```
$ oc exec -n metallb-system frr-k8s-thsmw -c frr -- vtysh -c "show bgp ipv4 unicast
203.0.113.200/30"
```

Replace **ipv4** with **ipv6** to display the BGP peers that received an IPv6 address pool. Replace **203.0.113.200/30** with an IPv4 or IPv6 IP address range from an address pool.

Example output

```
BGP routing table entry for 203.0.113.200/30
Paths: (1 available, best #1, table default)
  Advertised to non peer-group peers:
  10.0.2.3
  Local
    0.0.0.0 from 0.0.0.0 (10.0.1.2)
      Origin IGP, metric 0, weight 32768, valid, sourced, local, best (First path received)
      Last update: Mon Jan 10 19:49:07 2022
```

where:

10.0.2.3

Specifies that the output includes an IP address for a BGP peer.

4.9.3. Troubleshooting BFD issues

To diagnose and resolve Bidirectional Forwarding Detection (BFD) issues, execute commands directly within the FRRouting (FRR) container. By accessing the container, you can verify that BFD peers are correctly configured with established BGP sessions.

The BFD implementation that Red Hat supports uses FRRouting (FRR) in a container that exists in a **speaker** pod.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Display the names of the **speaker** pods:

```
$ oc get -n metallb-system pods -l component=speaker
```

Example output

```
NAME            READY   STATUS    RESTARTS   AGE
speaker-66bth   4/4     Running   0          26m
speaker-gvfnf   4/4     Running   0          26m
...
```

2. Display the BFD peers:

```
$ oc exec -n metallb-system speaker-66bth -c frr -- vtysh -c "show bfd peers brief"
```

Example output

```
Session count: 2
SessionId  LocalAddress             PeerAddress             Status
=========  ============             ===========             ======
3909139637 10.0.1.2                 10.0.2.3                up
```

where:

up

Specifies that the **PeerAddress** column includes each BFD peer. If the output does not list a BFD peer IP address that you expected the output to include, troubleshoot BGP connectivity with the peer. If the status field indicates **down**, check for connectivity on the links and equipment between the node and the peer. You can determine the node name for the speaker pod with a command like **oc get pods -n metallb-system speaker-66bth -o jsonpath='{.spec.nodeName}'**.

4.9.4. MetalLB metrics for BGP and BFD

To monitor network connectivity and diagnose routing states, refer to the Prometheus metrics for MetalLB. These metrics provide visibility into the status of BGP peers and BFD profiles so that you can ensure stable external communication.

Table 4.11. MetalLB BFD metrics

Name	Description
frrk8s_bfd_control_packet_input	Counts the number of BFD control packets received from each BFD peer.
frrk8s_bfd_control_packet_output	Counts the number of BFD control packets sent to each BFD peer.

Name	Description
frrk8s_bfd_echo_packet_input	Counts the number of BFD echo packets received from each BFD peer.
frrk8s_bfd_echo_packet_output	Counts the number of BFD echo packets sent to each BFD.
frrk8s_bfd_session_down_events	Counts the number of times the BFD session with a peer entered the **down** state.
frrk8s_bfd_session_up	Indicates the connection state with a BFD peer. **1** indicates the session is **up** and **0** indicates the session is **down**.
frrk8s_bfd_session_up_events	Counts the number of times the BFD session with a peer entered the **up** state.
frrk8s_bfd_zebra_notifications	Counts the number of BFD Zebra notifications for each BFD peer.

Table 4.12. MetalLB BGP metrics

Name	Description
frrk8s_bgp_announced_prefixes_total	Counts the number of load balancer IP address prefixes that are advertised to BGP peers. The terms *prefix* and *aggregated route* have the same meaning.
frrk8s_bgp_session_up	Indicates the connection state with a BGP peer. **1** indicates the session is **up** and **0** indicates the session is **down**.
frrk8s_bgp_updates_total	Counts the number of BGP update messages sent to each BGP peer.
frrk8s_bgp_opens_sent	Counts the number of BGP open messages sent to each BGP peer.
frrk8s_bgp_opens_received	Counts the number of BGP open messages received from each BGP peer.
frrk8s_bgp_notifications_sent	Counts the number of BGP notification messages sent to each BGP peer.
frrk8s_bgp_updates_total_received	Counts the number of BGP update messages received from each BGP peer.
frrk8s_bgp_keepalives_sent	Counts the number of BGP keepalive messages sent to each BGP peer.

Name	Description
frrk8s_bgp_keepalives_received	Counts the number of BGP keepalive messages received from each BGP peer.
frrk8s_bgp_route_refresh_sent	Counts the number of BGP route refresh messages sent to each BGP peer.
frrk8s_bgp_total_sent	Counts the number of total BGP messages sent to each BGP peer.
frrk8s_bgp_total_received	Counts the number of total BGP messages received from each BGP peer.

Additional resources

- Querying metrics for all projects with the monitoring dashboard

4.9.5. About collecting MetalLB data

To collect diagnostic data for debugging or support analysis, execute the **oc adm must-gather** CLI command. This utility captures essential information regarding the cluster, the MetalLB configuration, and the MetalLB Operator state.

The following list details features and objects related to MetalLB and the MetalLB Operator:

- The namespace and child objects where you deploy the MetalLB Operator
- All MetalLB Operator custom resource definitions (CRDs)

The command collects the following information from FRRouting (FRR), which Red Hat uses to implement BGP and BFD:

- **/etc/frr/frr.conf**
- **/etc/frr/frr.log**
- **/etc/frr/daemons** configuration file
- **/etc/frr/vtysh.conf**

The command collects log and configuration files from the **frr** container that exists in each **speaker** pod. Additionally, the command collects the output from the following **vtysh** commands:

- **show running-config**
- **show bgp ipv4**
- **show bgp ipv6**
- **show bgp neighbor**
- **show bfd peer**

No additional configuration is required when you run the command.

Additional resources

- Gathering data about your cluster