

# Networking

## Guides

### Configure Domain

Example Domain custom resource (CR)

Creating Domain by using the web console

Creating Domain by using the CLI

Subsequent Actions

Additional resources

### Creating Certificates

Creating a certificate by using the web console

### Configure Ingresses

Implementation Method

Example Ingress:

Creating an Ingress by using the CLI

### Configure Service

Why Service is needed

Example ClusterIP Service

Headless Service

### Configure Subnets

IP Allocation Rules

Calico Network

Kube-OVN Network

Subnet Management

### Configure MetalLB

Prerequisites

Configure an External IP Address Pool by using the web console

Configure BGP Peers by using the web console

Configure an External IP Address Pool with L2Advertisement

### Configure GatewayAPI Gateway

Prerequisites

Configuration Via Web Console

Configuration Via YAML

Introduction

### Configure Gateway

Prerequisites

Configuration Via Web Console

[Next Step](#)[Configuration V](#)[Introduction](#)

## Configure AI B

### Configure NodeLocal DNSCache

[Overview](#)[Key Features](#)[Important Notes](#)[Installation](#)[How It Works](#)[Configuration](#)

## Configure C

[Overview](#)[Configuration](#)

## How To

### Tasks for Ingress-Nginx

[Prerequisites](#)[Max Connections](#)[Request Timeout](#)[Session Affinity \(Sticky Sessions\)](#)[Header Modification](#)[URL Rewrite](#)[HSTS \(HTTP Strict Transport Security\)](#)[Rate Limiting](#)[WAF](#)[Forward-header control](#)[HTTPS](#)[Preserve Source IP](#)

### Tasks for Envoy Gateway

[Prerequisites](#)[Introduction](#)[Common Tasks For Route Config](#)[Advanced Configuration](#)[More configuration](#)

### Configure Endpoint Health Che

[Overview](#)[Key Features](#)[Installation](#)[How It Works](#)[How To Activate](#)

### Soft Data C

[Prerequisites](#)[Procedure](#)[Verification](#)

### Kube OVN

[alb](#)

Uninstallation

---

## Trouble Shooting

[How to Solve Inter-node Comm](#) [Find Who Cause the Error](#)

# Guides

## Configure Domain

Example Domain custom resource (CR)  
 Creating Domain by using the web console  
 Creating Domain by using the CLI  
 Subsequent Actions  
 Additional resources

## Creating Certificates

Creating a certificate by using the web console

## Configure Ingresses

Implementation Method  
 Example Ingress:  
 Creating a Ingress by using the CLI

## Configure Service

Why Service is needed  
 Example Cluster Service  
 Headless Service

## Configure Subnets

IP Allocation Rules  
 Calico Network  
 Kube-OVN Network  
 Subnet Management

## Configure MetalLB

Prerequisites  
 Configure an External IP Address Pool by using the web console  
 Configure BGP Peers by using the web console  
 Configure an External IP Address Pool with L2Advertisement

## Configure GatewayAPI Gateway

Prerequisites  
 Configuration Via Web Console  
 Configuration Via YAML  
 Introduction  
 Next Step

## Configure Gateway

Prerequisites  
 Configuration Via Web Console  
 Configuration Via YAML  
 Introduction

## Configure ALB

### Configure NodeLocal DNSCache

Overview

Key Features

Important Notes

Installation

How It Works

Configuration

### Configure C

Overview

Configuration

# Configure Domain

Add domain name resources to the platform and allocate domains for use by all projects under a cluster or resources under a specific project. When creating a domain name, binding a certificate is supported.

## NOTE

The domain names created on the platform should be resolved to the cluster's load balancing address before they can be accessed via the domain name. Therefore, you need to ensure that the domain names added on the platform have been successfully registered and that the domain names resolve to the cluster's load balancing address.

Successfully created and allocated domain names on the platform can be utilized in the following features of **Container Platform**:

- **Create Inbound Rules: Network Management > Inbound Rules > Create Inbound Rule**
- **Create Native Applications: Application Management > Native Applications > Create Native Application > Add Inbound Rule**
- **Add Listening Ports for Load Balancing: Network Management > Load Balancer Details > Add Listening Port**

Once the domain name is bound to a certificate, application developers can simply select the domain name when configuring the load balancer and inbound rules, allowing the use of the certificate that comes with the domain name for https support.

---

## TOC

Creating Domain by using the web console

Creating Domain by using the CLI

Subsequent Actions

Additional resources

---

## Example Domain custom resource (CR)

```
# test-domain.yaml
apiVersion: crd.alauda.io/v2
kind: Domain
metadata:
  name: '00000000003075575260129686e67ed4-917a-454a-8553-d55fc4030f81'
  annotations:
    cpaas.io/secret-ref: developer.test.cn-xfd8x 1
  labels:
    cluster.cpaas.io/name: global
    project.cpaas.io/name: cong
spec:
  name: developer.test.cn
  kind: full
```

<sup>1</sup> If certificates are enabled, an LTS-type Secret must be created in advance. The `secret-ref` is secret name.

## Creating Domain by using the web console

1. Go to **Administrator**.
2. In the left navigation bar, click **Network Management > Domain Names**.
3. Click **Create Domain Name**.
4. Configure the relevant parameters according to the following instructions.

Parameter	Description
Type	<ul style="list-style-type: none"> <li>Domain: A complete domain name, e.g., <code>developer.test.cn</code>.</li> <li>Wildcard Domain: A wildcard domain with a wildcard (*) character, e.g., <code>*.test.cn</code>, which includes all subdomains under the domain <code>test.cn</code>.</li> </ul>
Domain	Enter a complete domain name or domain suffix based on the selected domain name type.
Allocate Cluster	If a cluster is allocated, you also need to select a project associated with the allocated cluster, such as all projects associated with the cluster.
Certificate	<p>Includes the public key (tls.crt) and private key (tls.key) for creating a domain name-bound certificate. The project to which the certificate is allocated is the same as the bound domain name.</p> <p><b>Notes:</b></p> <ul style="list-style-type: none"> <li>Binary file imports are not supported.</li> <li>The bound certificate should meet the conditions of correct format, within the validity period, and signed for the domain name, etc.</li> <li>After creating the bound certificate, the name format of the bound certificate is: domain name - random characters.</li> <li>After creating the bound certificate, the bound certificate can be viewed in the certificate list, but updates and deletions of the bound certificate are only supported on the domain detail page.</li> <li>After creating the bound certificate, updating the certificate content is supported, but replacing other certificates is not supported.</li> </ul>

5. Click **Create**.

## Creating Domain by using the CLI

```
kubectl apply -f test-domain.yaml
```

## Subsequent Actions

- **Domain Registration:** Register the domain if the created domain has not been registered.
- **Domain Resolution:** Perform domain resolution if the domain does not point to the platform cluster's load balancing address.

## Additional resources

- [Configure Certificate](#)

# Creating Certificates

After the platform administrator imports the TLS certificate and assigns it to a specified project, developers with corresponding project permissions can use the certificate imported and assigned by the platform administrator when using inbound rules and load balancing functionalities. Subsequently, in scenarios such as certificate expiration, the platform administrator can update the certificate centrally.

## NOTE

The certificate functionality is currently not supported for use in public cloud clusters. You can create TLS type secret dictionaries as needed within the specified namespace.

## TOC

[Creating a certificate by using the web console](#)

## Creating a certificate by using the web console

1. Go to **Administrator**.
2. In the left navigation bar, click **Network Management > Certificates**.
3. Click **Create Certificate**.
4. Refer to the instructions below to configure the relevant parameters.

Parameter	Description
<b>Assign Project</b>	<ul style="list-style-type: none"><li>• All Projects: Assign the certificate for use in all projects associated with the current cluster.</li><li>• Specified Project: Assign the certificate for use in the specified project.</li><li>• No Assignment: Do not assign a project for now. After the certificate creation is completed, you can update the projects that can use the certificate through the <b>Update Project</b> operation.</li></ul>
<b>Public Key</b>	This refers to tls.crt. When importing the public key, binary files are not supported.
<b>Private Key</b>	This refers to tls.key. When importing the private key, binary files are not supported.

5. Click **Create**.

# Configure Services

In Kubernetes, a Service is a method for exposing a network application that is running as one or more Pods in your cluster.

## TOC

### [Why Service is Needed](#)

Example ClusterIP type Service:

Headless Services

Creating a service by using the web console

Creating a service by using the CLI

Example: Accessing an Application Within the Cluste

Example: Accessing an Application Outside the Cluste

Example: ExternalName type of Service

LoadBalancer Type Service Annotations

AWS EKS Cluster

Huawei Cloud CCE Cluster

Azure AKS Cluster

Google GKE Cluster

Example: LoadBalancer with MetallB BGP and Local Traffic Policy

Benefits

Prerequisites

Steps

Key Configuration Points

externalTrafficPolicy: Local

LoadBalancer with BGP

Deployment Steps

Verification

## Why Service is Needed

1. Pods have their own IPs, but:

- Pod IPs are not stable (they change if the Pod is recreated).
- Directly accessing Pods becomes unreliable.

2. Service solves this by providing:

- A stable IP and DNS name.
- Automatic load balancing to the matching Pods.

## Example ClusterIP type Service:

```
# simple-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  type: ClusterIP ①
  selector: ②
    app.kubernetes.io/name: MyApp
  ports:
    - protocol: TCP
      port: 80 ③
      targetPort: 80 ④
```

① The available type values and their behaviors are `ClusterIP`, `NodePort`, `LoadBalancer`, `ExternalName`

- 2 The set of Pods targeted by a Service is usually determined by a selector that you define.
- 3 `Service` port.
- 4 Bind `targetPort` of the Service to the Pod `containerPort`. In addition, you can reference `port.name` under the pod container.

## Headless Services

Sometimes you don't need load-balancing and a single Service IP. In this case, you can create what are termed headless Services:

```
spec:
  clusterIP: None
```

Headless Services are useful when:

- You want to discover individual Pod IPs, not just a single service IP.
- You need direct connections to each Pod (e.g., for databases like Cassandra or StatefulSets).
- You're using StatefulSets where each Pod must have a stable DNS name.

## Creating a service by using the web console

1. Go to **Container Platform**.
2. In the left navigation bar, click **Network > Services**.
3. Click **Create Service**.
4. Refer to the following instructions to configure the relevant parameters.

Parameter	Description
<b>Virtual IP Address</b>	If enabled, a ClusterIP will be allocated for this Service, which can be used for service discovery within the cluster.

Parameter	Description
	<p>If disabled, a Headless Service will be created, which is usually used by <b>StatefulSet</b>.</p>
<p><b>Type</b></p>	<ul style="list-style-type: none"> <li>• <b>ClusterIP</b>: Exposes the Service on a cluster-internal IP. Choosing this value makes the Service only reachable from within the cluster.</li> <li>• <b>NodePort</b>: Exposes the Service on each Node's IP at a static port (the NodePort).</li> <li>• <b>ExternalName</b>: Maps the Service to the contents of the externalName field (for example, to the hostname api.foo.bar.example).</li> <li>• <b>LoadBalancer</b>: Exposes the Service externally using an external load balancer. Kubernetes does not directly offer a load balancing component; you must provide one, or you can integrate your Kubernetes cluster with a cloud provider.</li> </ul>
<p><b>Target Component</b></p>	<ul style="list-style-type: none"> <li>• <b>Workload</b>: The Service will forward requests to a <b>specific</b> workload, which matches the labels like <code>project.cpaas.io/name: projectname</code> and <code>service.cpaas.io/name: deployment-name</code>.</li> <li>• <b>Virtualization</b>: The Service will forward requests to a <b>specific</b> virtual machine or virtual machine group.</li> <li>• <b>Label Selector</b>: The Service will forward requests to a <b>certain type</b> of workload with specified labels, for example, <code>environment: release</code>.</li> </ul>
<p><b>Port</b></p>	<p>Used to configure the port mapping for this Service. In the following example, other pods within the cluster can call this Service via the virtual IP (if enabled) and TCP port 80; the access requests will be forwarded to the externally exposed TCP port 6379 or <i>redis</i> of the target component's pods.</p> <ul style="list-style-type: none"> <li>• <b>Protocol</b>: The protocol used by the Service, supported protocols include: <code>TCP</code>, <code>UDP</code>, <code>HTTP</code>, <code>HTTP2</code>, <code>HTTPS</code>, <code>gRPC</code>.</li> </ul>

Parameter	Description
	<ul style="list-style-type: none"> <li>• <b>Service Port:</b> The service port number exposed by the Service within the cluster, that is, Port, e.g., 80.</li> <li>• <b>Container Port:</b> The target port number (or name) that the service port maps to, that is, targetPort, e.g., 6379 or <i>redis</i>.</li> <li>• <b>Service Port Name:</b> Will be generated automatically. The format is <code>&lt;protocol&gt;-&lt;service port&gt;-&lt;container port&gt;</code>, for example: <i>tcp-80-6379</i> or <i>tcp-80-redis</i>.</li> </ul>
<b>Session Affinity</b>	Session affinity based on the source IP address (ClientIP). If enabled, all access requests from the same IP address will be kept on the same server during load balancing, ensuring that requests from the same client are forwarded to the same server for processing.

5. Click **Create**.

## Creating a service by using the CLI

```
kubectl apply -f simple-service.yaml
```

Create a service based on an existing deployment resource `my-app`.

```
kubectl expose deployment my-app \
  --port=80 \
  --target-port=8080 \
  --name=test-service \
  --type=NodePort \
  -n p1-1
```

## Example: Accessing an Application Within the Cluste

```
# access-internal-demo.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.25
          ports:
            - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: nginx-clusterip
spec:
  type: ClusterIP
  selector:
    app: nginx
  ports:
    - port: 80
      targetPort: 80
```

### 1. Apply this YAML:

```
kubectl apply -f access-internal-demo.yaml
```

### 2. Starting another Pod:

```
kubectl run test-pod --rm -it --image=busybox -- /bin/sh
```

### 3. Accessing the `nginx-clusterip` service in `test-pod` Pod:

```
wget -qO- http://nginx-clusterip
# or using DNS records created automatically by Kubernetes: <service-name>.<namespace>.svc.cluster.local
wget -qO- http://nginx-clusterip.default.svc.cluster.local
```

You should see a HTML response containing text like "Welcome to nginx!".

## Example: Accessing an Application Outside the Cluste

```
# access-external-demo.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.25
          ports:
            - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: nginx-nodeport
spec:
  type: NodePort
  selector:
    app: nginx
  ports:
    - port: 80
      targetPort: 80
      nodePort: 30080
```

## 1. Apply this YAML:

```
kubectl apply -f access-external-demo.yaml
```

## 2. Checking Pods:

```
kubectl get pods -l app=nginx -o wide
```

### 3. curl Service:

```
curl http://{NodeIP}:{nodePort}
```

You should see a HTML response containing text like "Welcome to nginx!".

Of course, it is also possible to access the application from outside the cluster by creating a Service of type LoadBalancer.

**Note:** Please configure the LoadBalancer service beforehand.

```
# access-external-demo-with-loadbalancer.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.25
          ports:
            - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: nginx-lb-service
spec:
  type: LoadBalancer
  selector:
    app: nginx
  ports:
    - port: 80
      targetPort: 80
```

1. Apply this YAML:

```
kubectl apply -f access-external-demo-with-loadbalancer.yaml
```

2. Get external ip address:

```
kubectl get svc nginx-lb-service
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)
nginx-service	LoadBalancer	10.0.2.57	34.122.45.100	80:3000
AGE				
5/TCP	30s			

`EXTERNAL-IP` is the address you access from your browser.

```
curl http://34.122.45.100
```

You should see a HTML response containing text like "Welcome to nginx!".

If `EXTERNAL-IP` is `pending`, the Loadbalancer service is not currently deployed on the cluster.

## Example: ExternalName type of Service

```
apiVersion: v1
kind: Service
metadata:
  name: my-external-service
  namespace: default
spec:
  type: ExternalName
  externalName: example.com
```

1. Apply this YAML:

```
kubectl apply -f external-service.yaml
```

2. Try to resolve inside a Pod in the cluster:

```
kubectl run test-pod --rm -it --image=busybox -- sh
```

then:

```
nslookup my-external-service.default.svc.cluster.local
```

You'll see that it resolves to `example.com`.

## LoadBalancer Type Service Annotations

### AWS EKS Cluster

For detailed explanations of the EKS LoadBalancer Service annotations, please refer to the [Annotation Usage Documentation](#).

Key	Value	Description
<code>service.beta.kubernetes.io/aws-load-balancer-type</code>	external: Use the official AWS LoadBalancer Controller.	Specifies the controller for the LoadBalancer type.  <b>Note:</b> Please contact the platform administrator in advance to deploy the AWS LoadBalancer Controller.
<code>service.beta.kubernetes.io/aws-load-balancer-nlb-target-type</code>	<ul style="list-style-type: none"> <li>instance: Traffic will be sent to the pods via NodePort.</li> <li>ip: Traffic routes directly to the pods (the cluster must use Amazon VPC CNI).</li> </ul>	Specifies how traffic reaches the pods.
<code>service.beta.kubernetes.io/aws-load-balancer-scheme</code>	<ul style="list-style-type: none"> <li>internal: Private network.</li> </ul>	Specifies whether to use a private network or a public network.

Key	Value	Description
	<ul style="list-style-type: none"> <li>internet-facing: Public network.</li> </ul>	
service.beta.kubernetes.io/aws-load-balancer-ip-address-type	<ul style="list-style-type: none"> <li>IPv4</li> <li>dualstack</li> </ul>	Specifies the supported IP address stack.

## Huawei Cloud CCE Cluster

For detailed explanations of the CCE LoadBalancer Service annotations, please refer to the [Annotation Usage Documentation](#) .

Key	Value
kubernetes.io/elb.id	
kubernetes.io/elb.autocreate	<p>Example: <code>{"type":"public","bandwidth_name":"cce-bandwidth-1551163379627","bandwidth_chargemode":"bandwidth","bandwidth_flavor_name":["cn-north-4b"],"l4_flavor_name":"L4_flavor.elb.s1.small"}</code></p> <p><b>Note:</b> Please read the <a href="#">Filling Instructions</a> first and adjust the exam</p>
kubernetes.io/elb.subnet-id	

Key	Value
kubernetes.io/elb.class	<ul style="list-style-type: none"><li>• union: Shared load balancing.</li><li>• performance: Exclusive load balancing, only supported in Kuberr</li></ul>
kubernetes.io/elb.enterpriseID	

## Azure AKS Cluster

For detailed explanations of the AKS LoadBalancer Service annotations, please refer to the [Annotation Usage Documentation](#) .

Key	Value	Description
service.beta.kubernetes.io/azure-load-balancer-internal	<ul style="list-style-type: none"> <li>true: Private network.</li> <li>false: Public network.</li> </ul>	Specifies whether to use a private network or a public network.

## Google GKE Cluster

For detailed explanations of the GKE LoadBalancer Service annotations, please refer to the [Annotation Usage Documentation](#) .

Key	Value	Description
networking.gke.io/load-balancer-type	Internal	Specifies the use of a private network.
cloud.google.com/I4-rbs	enabled	Defaults to public. If this parameter is configured, traffic will route directly to the pods.

## Example: LoadBalancer with MetalLB BGP and Local Traffic Policy

This example demonstrates how to configure a LoadBalancer Service using MetalLB BGP mode with `externalTrafficPolicy: Local` to achieve active-active load balancing without extra network hops.

### Benefits

- **Active-active load balancing:** Traffic is distributed across multiple nodes simultaneously

- **No extra network hops:** Direct routing to pods without intermediate node forwarding
- **Better performance:** `externalTrafficPolicy: Local` preserves source IP and reduces latency
- **High availability:** BGP route announcements ensure traffic reaches healthy nodes

## Prerequisites

Before configuring the LoadBalancer Service, ensure you have:

1. **MetallB deployed:** See [Creating External IP Address Pool](#) for installation
2. **BGP Peer configured:** See [Creating BGP Peers](#) for BGP setup
3. **External IP address pool:** Configure an IPAddressPool with BGPAdvertisement

## Steps

Deploy your application with a LoadBalancer Service using `externalTrafficPolicy: Local`:

```
# nginx-loadbalancer-local-demo.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.25
          ports:
            - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: nginx-loadbalancer-local
spec:
  type: LoadBalancer
  externalTrafficPolicy: Local
  selector:
    app: nginx
  ports:
    - port: 80
      targetPort: 80
```

## Key Configuration Points

### externalTrafficPolicy: Local

The `externalTrafficPolicy: Local` setting provides several benefits:

- **Source IP preservation:** Client source IP is maintained, enabling proper logging and security policies
- **Direct pod routing:** Traffic goes directly to pods without node-level forwarding

## LoadBalancer with BGP

When using MetalLB with BGP mode:

- Routes are advertised from nodes specified in the BGPAdvertisement nodeSelectors
- The BGP peer receives these announcements and can route traffic accordingly
- Node selector alignment between BGPPeer and BGPAdvertisement ensures consistent routing

## Deployment Steps

### 1. Deploy the application:

```
kubectl apply -f nginx-loadbalancer-local-demo.yaml
```

### 2. Verify the LoadBalancer Service:

```
kubectl get svc nginx-loadbalancer-local
```

Expected output:

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	P
nginx-loadbalancer-local	LoadBalancer	10.0.2.57	4.4.4.3	8
0:30005/TCP	30s			

### 3. Test the service:

```
curl http://4.4.4.3
```

## Verification

- **Monitor service endpoints:** `kubectl get endpoints nginx-loadbalancer-local`
- **Check service status:** `kubectl describe svc nginx-loadbalancer-local`

# Configure Ingresses

Ingress rules (Kubernetes Ingress) expose HTTP/HTTPS routes from outside the cluster to internal routing (Kubernetes Service), enabling control of external access to computing components.

Create an Ingress to manage the external HTTP/HTTPS access to a Service.

## WARNING

When creating multiple ingresses within the same namespace, different ingresses **MUST NOT** have the same **Domain**, **Protocol**, and **Path** (i.e., duplicate access points are not allowed).

## TOC

### [Implementation Method](#)

Example Ingress:

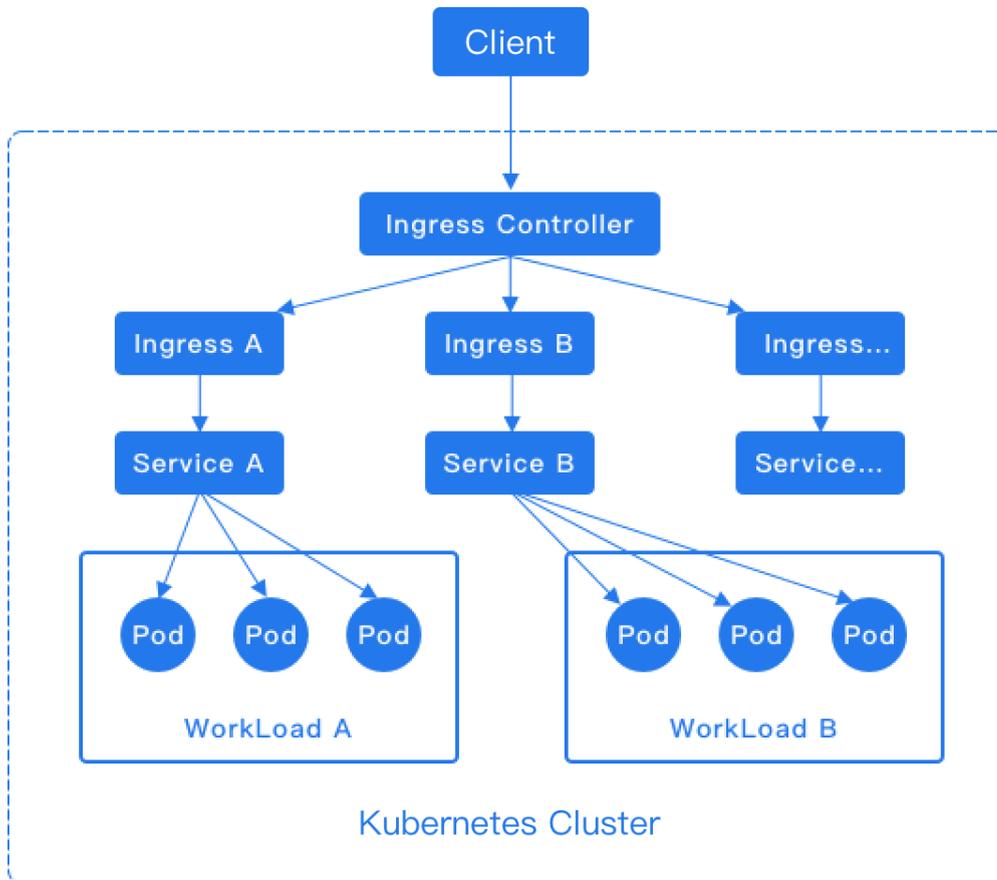
[Creating a Ingress by using the web console](#)

[Creating a Ingress by using the CLI](#)

## Implementation Method

Ingress rules depend on the implementation of the Ingress Controller, which is responsible for listening to changes in Ingress and Service. After a new Ingress is created, when the Ingress

Controller receives a request, it matches the forwarding rule from the Ingress and distributes the traffic to the specified internal routes, as shown in the diagram below.



#### NOTE

For the HTTP protocol, Ingress only supports the 80 port as the external port. For the HTTPS protocol, Ingress only supports the 443 port as the external port. The platform's load balancer will automatically add the 80 and 443 listening ports.

- [Install ingress-nginx as ingress-controller via ingress-nginx-operator](#)
- [Install alb as ingress-controller via alb-operator](#)

## Example Ingress:

```
# nginx-ingress.yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: nginx-ingress
  namespace: k-1
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: / ❶
spec:
  ingressClassName: nginx ❷
  rules:
    - host: demo.local ❸
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: nginx-service
                port:
                  number: 80
```

- ❶ To see more configurations please refer to [nginx-configuration](#).
- ❷ `nginx` to using `ingress-nginx` controller, `$alb_name` to use alb as ingress controller.
- ❸ If you only want to run ingress locally, configure the `hosts` beforehand.

## Creating a Ingress by using the web console

1. Access the **Container Platform**.
2. In the left navigation bar, click **Network > Ingress**.
3. Click **Create Ingress**.
4. Reference the instructions below to configure certain parameters.

Parameter	Description
<b>Ingress Class</b>	Ingresses can be implemented by different controllers with different <code>IngressClass</code> name. If multiple ingress controllers are available on the platform, the user can select which one to use with this option.
<b>Domain Name</b>	Hosts can be precise matches (for example <code>foo.bar.com</code> ) or a wildcard (for example <code>*.foo.com</code> ). The domain names available are allocated by platform administrator.
<b>Certificates</b>	TLS secret or Certificates allocated by platform administrator.
<b>Match Type and Path</b>	<ul style="list-style-type: none"> <li>• <b>Prefix:</b> Matches path prefixes, e.g., <code>/abcd</code> can match <code>/abcd/efg</code> or <code>/abcde</code>.</li> <li>• <b>Exact:</b> Matches exact paths, e.g., <code>/abcd</code>.</li> <li>• <b>Implementation specific:</b> If you are using a custom Ingress controller to manage the Ingress rules, you may choose to have the controller decide.</li> </ul>
<b>Service</b>	External traffic will be forwarded to this Service.
<b>Service Port</b>	Specify which Service port the traffic will be forwarded to.

5. Click **Create**.

## Creating a Ingress by using the CLI

```
kubectl apply -f nginx-ingress.yaml
```

# Configure Subnets

---

## TOC

### IP Allocation Rules

#### Calico Network

- Constraints and Limitations

- Example Subnet custom resource (CR) with Calico Network

- Creating a Subnet in the Calico network by using the web console

- Creating a Subnet in the Calico network by using the CLI

- Reference Content

#### Kube-OVN Network

- Example Subnet custom resource (CR) with Kube-OVN Overlay Network

- Creating a Subnet in the Kube-OVN Overlay Network by using the web console

- Creating a Subnet in the Kube-OVN Overlay Network by using the the CLI

- Underlay Network

- Usage Instructions

- Add Bridge Network by using the web console (Optional)

- Add Bridge Network by using the CLI

- Add VLAN by using the web console (Optional)

- Add VLAN by using the CLI

- Example Subnet custom resource (CR) with Kube-OVN Underlay Network

- Creating a Subnet in the Kube-OVN Underlay Network by using the web console

- Creating a Subnet in the Kube-OVN Underlay Network by using the CLI

- Related Operations

#### Subnet Management

---

Updating Gateway by using the web console

Updating Gateway by using the CLI

Updating Reserved IPs by using the web console

Updating Reserved IPs by using the CLI

Assigning Projects by using the web console

Assigning Projects by using the CLI

Assigning Namespaces by using the web console

Assigning Namespaces by using the CLI

Expanding Subnets by using the web console

Expanding Subnets by using the CLI

Managing Calico Networks

Delete Subnet by using the web console

Delete Subnet by using the CLI

---

## IP Allocation Rules

### NOTE

If a project or namespace is assigned multiple subnets, an IP address will be randomly selected from one of the subnets.

- Project Allocation:
  - If a project is not bound to a subnet, Pods in all namespaces under that project can only use IP addresses from the default subnet. If there are insufficient IP addresses in the default subnet, the Pods will not be able to start.
  - If a project is bound to a subnet, Pods in all namespaces under that project can only use IP addresses from that specific subnet.
- Namespace Allocation:

- If a namespace is not bound to a subnet, Pods in that namespace can only use IP addresses from the default subnet. If there are insufficient IP addresses in the default subnet, the Pods will not be able to start.
- If a namespace is bound to a subnet, Pods in that namespace can only use IP addresses from that specific subnet.

## Calico Network

Creating subnets in the Calico network to achieve finer granularity of network isolation for resources within the cluster.

## Constraints and Limitations

In an IPv6 cluster environment, the subnets created within the Calico network, by default, use VXLAN encapsulation. The ports required for VXLAN encapsulation differ from those of IPIP encapsulation. You need to ensure that UDP port 4789 is open.

## Example Subnet custom resource (CR) with Calico Network

```
# test-calico-subnet.yaml
apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
  name: test-calico
spec:
  cidrBlock: 10.1.1.1/24
  default: false ①
  ipipMode: Always ②
  natOutgoing: true ③
  private: false
  protocol: Dual
  v4blockSize: 30
```

- ① When `default` If true, use VXLAN encapsulation.

- 2 See Encapsulation Mode parameters and Encapsulation Protocol parameters.
- 3 See Outbound Traffic NAT parameters.

## Creating a Subnet in the Calico network by using the web console

1. Go to **Administrator**.
2. In the left navigation bar, click **Network Management > Subnets**.
3. Click **Create Subnet**.
4. Refer to the following instructions to configure the relevant parameters.

Parameter	Description
<b>CIDR</b>	<p>After allocating the subnet to a project or namespace, the container groups within the namespace will randomly use IP addresses within this CIDR for communication.</p> <p><b>Note:</b> For the correspondence between CIDR and BlockSize, please refer to <a href="#">Reference Content</a>.</p>
<b>Encapsulation Protocol</b>	<p>Select the encapsulation protocol. <b>IPIP</b> is not supported in dual-stack mode.</p> <ul style="list-style-type: none"> <li>• <b>IPIP:</b> Implements inter-segment communication using the IPIP protocol.</li> <li>• <b>VXLAN (Alpha):</b> Implements inter-segment communication using the VXLAN protocol.</li> <li>• <b>No Encapsulation:</b> Directly connected through routing forwarding.</li> </ul>

Parameter	Description
<b>Encapsulation Mode</b>	<p>When the encapsulation protocol is <b>IPIP</b> or <b>VXLAN</b>, the encapsulation mode must be set, defaulting to <b>Always</b>.</p> <ul style="list-style-type: none"> <li>• <b>Always</b>: Always enable IPIP / VXLAN tunnels.</li> <li>• <b>Cross Subnet</b>: Enable IPIP / VXLAN tunnels only when the host is in different subnets; direct connection via routing forwarding when the host is in the same subnet.</li> </ul>
<b>Outbound Traffic NAT</b>	<p>Choose whether to enable outbound traffic NAT (Network Address Translation), which is enabled by default.</p> <p>It is primarily used to set the access addresses exposed to the external network when the subnet container group accesses the external network.</p> <p>When outbound traffic NAT is enabled, the host IP will be used as the access address for the current subnet container group; when not enabled, the IPs of the container groups in the subnet will be directly exposed to the external network.</p>

5. Click **Confirm**.

6. On the subnet details page, select **Actions** > **Allocate Project** / **Allocate Namespace**.

7. Complete the configuration and click **Allocate**.

## Creating a Subnet in the Calico network by using the CLI

```
kubectl apply -f test-calico-subnet.yaml
```

## Reference Content

The dynamic matching relationship between CIDR and blockSize is shown in the table below.

CIDR	blockSize Size	Number of Hosts	Size of a Single IP Pool
prefix<=16	26	1024+	64
16<prefix<=19	27	256~1024	32
prefix=20	28	256	16
prefix=21	29	256	8
prefix=22	30	256	4
prefix=23	30	128	4
prefix=24	30	64	4
prefix=25	30	32	4
prefix=26	31	32	2
prefix=27	31	16	2
prefix=28	31	8	2
prefix=29	31	4	2
prefix=30	31	2	2
prefix=31	31	1	2

**NOTE**

Subnet configurations with prefixes greater than 31 are not supported.

## Kube-OVN Network

Creating a subnet in the Kube-OVN Overlay Network to achieve more granular network isolation of resources in the cluster.

**NOTE**

The platform has a built-in **join** subnet for communication between nodes and Pods; please avoid conflicts in network segments between **join** and newly created subnets.

## Example Subnet custom resource (CR) with Kube-OVN Overlay Network

```
# test-overlay-subnet.yaml
apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
  name: test-overlay-subnet
spec:
  default: false
  protocol: Dual
  cidrBlock: 10.1.0.0/23
  natOutgoing: true ①
  excludeIps: ②
  - 10.1.1.2
  gatewayType: distributed ③
  gatewayNode: '' ④
  private: false
  enableEcmp: false ⑤
```

- ① See Outbound Traffic NAT parameters.
- ② See Reserved IP parameters.
- ③ See Gateway Type parameters. The available values are `distributed` or `centralized`.
- ④ See Gateway Nodes parameters.
- ⑤ See ECMP parameters. Provided that you contact the administrator to enable the feature gate.

## Creating a Subnet in the Kube-OVN Overlay Network by using the web console

1. Go to **Administrator**.
2. In the left navigation bar, click on **Network Management > Subnet**.
3. Click on **Create Subnet**.
4. Refer to the following instructions to configure the related parameters.

Parameter	Description
<b>Network Segment</b>	After assigning the subnet to the project or namespace, IPs within this segment will be randomly allocated for use by Pods.
<b>Reserved IP</b>	The set reserved IP will not be automatically allocated. For example, it can be used as the IP address for computing components' <b>fixed IP</b> .
<b>Gateway Type</b>	<p>Select the type of gateway for the subnet to control the outbound traffic.</p> <ul style="list-style-type: none"> <li>- <b>Distributed</b>: Each host in the cluster can act as an outbound node for Pods on the current host, enabling distributed egress.</li> <li>- <b>Centralized</b>: All Pods in the cluster use one or more specific hosts as outbound nodes, facilitating external auditing and firewall control. Setting multiple centralized <b>gateway nodes</b> can achieve high availability.</li> </ul>
<b>ECMP (Alpha)</b>	<p>When choosing a <b>Centralized</b> gateway, the ECMP feature can be used. By default, the gateway operates in master-slave mode, with only the master gateway processing traffic. When enabling ECMP (Equal-Cost Multipath Routing), outbound traffic will be routed through multiple equal-cost paths to all available gateway nodes, thereby increasing the total throughput of the gateway.</p> <p><b>Note</b>: Please enable ECMP-related features in advance.</p>
<b>Gateway Nodes</b>	When using a <b>Centralized</b> gateway, select one or more specific hosts as gateway nodes.
<b>Outbound Traffic NAT</b>	<p>Choose whether to enable outbound traffic NAT (Network Address Translation). By default, it is enabled.</p> <p>It is mainly used to set the access address exposed to the external network when the Pods in the subnet access the internet.</p> <p>When outbound traffic NAT is enabled, the host IP will be used as the</p>

Parameter	Description
	access address for the Pods in the current subnet; when not enabled, the IPs of the Pods within the subnet will be directly exposed to the external network. In this case, using a centralized gateway is recommended.

5. Click **Confirm**.
6. On the subnet details page, select **Actions > Allocate Project / Namespace**.
7. Complete the configuration and click **Allocate**.

## Creating a Subnet in the Kube-OVN Overlay Network by using the the CLI

```
kubectl apply -f test-overlay-subnet.yaml
```

## Underlay Network

Creating subnets in the Kube-OVN Underlay network not only enables finer-grained network isolation for resources but also provides a better performance experience.

### INFO

The container network in Kube-OVN Underlay requires support from the physical network. Please refer to the best practices [Preparing the Kube-OVN Underlay Physical Network](#) to ensure network connectivity.

## Usage Instructions

The general process for creating subnets in the Kube-OVN Underlay network is: Add Bridge Network > Add VLAN > Create Subnet.

- 1 Default Network Card Name.
- 2 Configure Network Card by Node.

## Add Bridge Network by using the web console (Optional)

```
# test-provider-network.yaml
kind: ProviderNetwork
apiVersion: kubeovn.io/v1
metadata:
  name: test-provider-network
spec:
  defaultInterface: eth1 ①
  customInterfaces: ②
  - interface: eth2
    nodes:
      - node1
  excludeNodes:
    - node2
```

- ① Default Network Card Name.
- ② Configure Network Card by Node.

A bridge network refers to a bridge, and after binding the network card to the bridge, it can forward container network traffic, achieving intercommunication with the physical network.

Procedure:

1. Go to **Administrator**.
2. In the left navigation bar, click **Network Management > Bridge Network**.
3. Click **Add Bridge Network**.
4. Configure the relevant parameters based on the following instructions.

**Note:**

- *Target Pod* refers to all Pods scheduled on the current node or Pods in namespaces bound to specific subnets scheduled to the current node. This depends on the scope of the subnet under the bridge network.
- The nodes in the Underlay subnet must have multiple network cards, and the network card used by the bridge network must be exclusively assigned to the Underlay and cannot carry other traffic, such as SSH. For example, if the bridge network has three nodes planning for eth0, eth0, eth1 for exclusive use by the Underlay, then the default network card can be set as eth0, and the network card for node three can be eth1.

Parameter	Description
<b>Default Network Card Name</b>	By default, the target Pod will use this as the bridge network card for intercommunication with the physical network.
<b>Configure Network Card by Node</b>	The target Pods on the configured nodes will bridge to the specified network card instead of the default network card.
<b>Exclude Nodes</b>	<p>When nodes are excluded, all Pods scheduled to these nodes will not bridge to any network card on these nodes.</p> <p><b>Note:</b> Pods on excluded nodes will not be able to communicate with the physical network or cross-node container networks, and care should be taken to avoid scheduling related Pods to these nodes.</p>

5. Click **Add**.

## Add Bridge Network by using the CLI

```
kubectl apply -f test-provider-network.yaml
```

## Add VLAN by using the web console (Optional)

```
# test-vlan.yaml
kind: Vlan
apiVersion: kubeovn.io/v1
metadata:
  name: test-vlan
spec:
  id: 0 ①
  provider: test-provider-network ②
```

- ① VLAN ID.
- ② Bridge network reference.

The platform has a pre-configured **ovn-vlan** virtual LAN, which will connect to the **provider** bridge network. You can also configure a new VLAN to connect to other bridge networks, thereby achieving network isolation between VLANs.

Procedure:

1. Navigate to **Administrator**.
2. In the left navigation bar, click **Network Management > VLAN**.
3. Click **Add VLAN**.
4. Configure the relevant parameters based on the following instructions.

Parameter	Description
<b>VLAN ID</b>	The unique identifier for this VLAN, which will be used to differentiate different virtual LANs.
<b>Bridge Network</b>	The VLAN will connect to this bridge network for intercommunication with the physical network.

5. Click **Add**.

## Add VLAN by using the CLI

```
kubectl apply -f test-vlan.yaml
```

## Example Subnet custom resource (CR) with Kube-OVN Underlay Network

```
# test-underlay-network.yaml
apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
  name: test-underlay-network
spec:
  default: false
  protocol: Dual
  cidrBlock: 11.1.0.0/23
  gateway: 11.1.0.1
  excludeIps:
    - 11.1.0.3
  private: false
  allowSubnets: []
  vlan: test-vlan ①
  enableEcmp: false
```

① VLAN reference.

## Creating a Subnet in the Kube-OVN Underlay Network by using the web console

### NOTE

The platform also pre-configures a **join** subnet for communication between nodes and Pods in Overlay transport mode. This subnet will not be used in Underlay transport mode, so it is crucial to avoid IP segment conflicts between **join** and other subnets.

Procedure:

1. Navigate to **Administrator**.
2. In the left navigation bar, click **Network Management > Subnet**.
3. Click **Create Subnet**.
4. Configure the relevant parameters based on the following instructions.

Parameter	Description
<b>VLAN</b>	The VLAN to which the subnet belongs.
<b>Subnet</b>	After assigning the subnet to a project or namespace, IPs within the physical subnet will be randomly allocated for use by Pods.
<b>Gateway</b>	The physical gateway within the above subnet.
<b>Reserved IP</b>	The specified reserved IP will not be automatically assigned. For example, it can be used as the IP for the compute component <b>fixed IP</b> .

5. Click **Confirm**.
6. On the subnet details page, select **Action > Assign Project / Namespace**.
7. Complete the configuration and click **Assign**.

## Creating a Subnet in the Kube-OVN Underlay Network by using the CLI

```
kubectl apply -f test-underlay-network.yaml
```

## Related Operations

When both Underlay and Overlay subnets exist in a cluster, you can configure the [Automatic Intercommunication Between Underlay and Overlay Subnets](#) as needed.

## Subnet Management

### Updating Gateway by using the web console

This includes changing the outbound traffic method, gateway nodes, and NAT configuration.

1. Go to **Administrator**.

2. In the left sidebar, click on **Network Management > Subnets**.
3. Click the name of the subnet.
4. Select **Action > Update Gateway**.
5. Update the parameter configurations; refer to the [Parameter Description](#) for details.
6. Click **OK**.

## Updating Gateway by using the CLI

```
kubectl patch subnet test-overlay-subnet --type=json -p='[
  {"op": "replace", "path": "/spec/gatewayType", "value": "centralized"},
  {"op": "replace", "path": "/spec/gatewayNode", "value": "192.168.66.21
0"},
  {"op": "replace", "path": "/spec/natOutgoing", "value": true},
  {"op": "replace", "path": "/spec/enableEcmp", "value": true}
]'
```

## Updating Reserved IPs by using the web console

The gateway IP cannot be removed from the reserved IPs, while other reserved IPs can be edited, deleted, or added freely.

1. Go to **Administrator**.
2. In the left sidebar, click on **Network Management > Subnets**.
3. Click the name of the subnet.
4. Select **Action > Update Reserved IP**.
5. After completing the updates, click **Update**.

## Updating Reserved IPs by using the CLI

```
kubectl patch subnet test-overlay-subnet --type=json -p='[
  {
    "op": "replace",
    "path": "/spec/excludeIps",
    "value": ["10.1.0.1", "10.1.1.2", "10.1.1.4"]
  }
]'
```

## Assigning Projects by using the web console

Assigning subnets to specific projects helps teams better manage and isolate network traffic for different projects, ensuring that each project has sufficient network resources.

1. Navigate to **Administrator**.
2. In the left sidebar, click on **Network Management > Subnets**.
3. Click the name of the subnet.
4. Select **Action > Assign Project**.
5. After adding or removing projects, click **Assign**.

## Assigning Projects by using the CLI

```
kubectl patch subnet test-overlay-subnet --type=json -p='[
  {
    "op": "replace",
    "path": "/spec/namespaceSelectors",
    "value": [
      {
        "matchLabels": {
          "cpaas.io/project": "cong"
        }
      }
    ]
  }
]'
```

## Assigning Namespaces by using the web console

Assigning subnets to specific namespaces allows for finer network isolation.

**Note:** The assignment process will rebuild the gateway, and outbound data packets will be discarded! Please ensure no business applications are currently accessing external clusters.

1. Navigate to **Administrator**.
2. In the left sidebar, click on **Network Management > Subnets**.
3. Click the name of the subnet.
4. Select **Action > Assign Namespace**.
5. After adding or removing namespaces, click **Assign**.

## Assigning Namespaces by using the CLI

```
kubectl patch subnet test-overlay-subnet --type=json -p='[
  {
    "op": "replace",
    "path": "/spec/namespaces",
    "value": ["cert-manager"]
  }
]'
```

## Expanding Subnets by using the web console

When the reserved IP range of a subnet reaches its usage limit or is about to be exhausted, it can be expanded based on the original subnet range without affecting the normal operation of existing services.

1. Navigate to **Administrator**.
2. In the left sidebar, click on **Network Management > Subnets**.
3. Click the name of the subnet.
4. Select **Action > Expand Subnet**.
5. Complete the configuration and click **Update**.

## Expanding Subnets by using the CLI

```
kubectl patch subnet test-overlay-subnet --type=json -p='[
  {
    "op": "replace",
    "path": "/spec/cidrBlock",
    "value": "10.1.0.0/22"
  }
]'
```

## Managing Calico Networks

Support for assigning projects and namespaces; for details, please refer to the [project assignment](#) and [namespace assignment](#).

## Delete Subnet by using the web console

### NOTE

- When a subnet is deleted, if there are still container groups using the IPs within the subnet, the container groups can continue to run and the IP addresses will remain unchanged, but they will be unable to communicate over the network. The container groups can be rebuilt to use IPs within the default subnet, or assign a new subnet to the namespace where the container groups reside for usage.
- The default subnet cannot be deleted.

1. Go to **Administrator**.
2. In the left navigation bar, click **Network Management > Subnets**.
3. Click **> Delete**, and proceed with the deletion.

## Delete Subnet by using the CLI

```
kubectl delete subnet test-overlay-subnet
```

# Configure MetalLB

---

## TOC

### Prerequisites

Configure an External IP Address Pool by using the web console

Configure BGP Peers by using the web console

Configure an External IP Address Pool with L2Advertisement or BGPAdvertisement by using the CLI

Troubleshooting MetalLB

---

## Prerequisites

Please ensure that you have read the [Installation](#) documentation before proceeding.

## Configure an External IP Address Pool by using the web console

1. Go to **Administrator**.
  2. In the left navigation bar, click **Network Management > External IP Address Pool**.
  3. Click **Create External IP Address Pool**.
  4. Refer to the following instructions to configure certain parameters.
-

Parameter	Description
Type	<ul style="list-style-type: none"> <li>L2: Communication and forwarding based on MAC addresses, suitable for small-scale or local area networks that require simple and fast layer 2 switching, with advantages in simple configuration and low latency.</li> <li>BGP (Alpha): Routing and forwarding based on IP addresses, using BGP protocol to exchange routing information, suitable for large-scale networks requiring complex routing across multiple autonomous systems, with advantages in high scalability and reliability.</li> </ul>
IP Resources	<p>Support input in CIDR and IP range formats. Click <b>Add</b> to support multiple entries, examples as follows:</p> <p><b>CIDR:</b> <input type="text" value="192.168.1.1/24"/> .</p> <p><b>IP Range:</b> <input type="text" value="192.168.2.1"/> ~ <input type="text" value="192.168.2.255"/> .</p>
Available Nodes	<p>In L2 mode, available nodes are those used to carry all VIP traffic; in BGP mode, available nodes are those used to carry VIPs, establish BGP connections with peers, and announce routes externally.</p> <ul style="list-style-type: none"> <li><b>Node Name:</b> Select available nodes based on node names.</li> <li><b>Label Selector:</b> Select available nodes based on labels.</li> <li><b>Show Node Details:</b> View final available nodes in a list format.</li> </ul> <p><b>Note:</b></p> <ul style="list-style-type: none"> <li>When using BGP type, the available nodes are the next-hop nodes; ensure that the selected available nodes are a subset of the BGP Connection Nodes.</li> <li>You can configure either the label selector or the node name separately to choose available nodes; if both are configured simultaneously, the final available nodes are the intersection of both.</li> </ul>
BGP Peers	Select BGP peers; please refer to <a href="#">BGP Peers</a> for specific configurations.

5. Click **Create**.

# Configure BGP Peers by using the web console

1. Go to **Administrator**.
2. In the left navigation bar, click **Network Management > BGP Peers**.
3. Click **Create BGP Peer**.
4. Refer to the instructions below to configure the parameters.

Parameter	Description
<b>Local AS Number</b>	<p>The AS number of the AS where the BGP-connected node resides.</p> <p><b>Note:</b> If there are no special requirements, it is recommended to use an IBGP configuration, meaning the local AS number should be consistent with the peer AS number.</p>
<b>Peer AS Number</b>	The AS number of the AS where the BGP peer resides.
<b>Peer IP</b>	The IP address of the BGP peer, which must be a valid IP address capable of establishing a BGP connection.
<b>Local IP</b>	The IP address of the BGP-connected node. When the BGP-connected node has multiple IPs, select the specified local IP to establish a BGP connection with the peer.
<b>Peer Port</b>	The port number of the BGP peer.
<b>BGP-Connected Node</b>	The node that establishes the BGP connection. If this parameter is not configured, all nodes will establish BGP connections.
<b>eBGP Multi-Hop</b>	Allows the establishment of BGP sessions between BGP routers that are not directly connected. When this feature is enabled, the default TTL value of BGP packets is 5, allowing the establishment of BGP peer relationships across multiple intermediate network devices, making network design more flexible.

Parameter	Description
<b>RouterID</b>	A 32-bit numeric value (usually represented in dotted-decimal format, similar to IPv4 address format) used to uniquely identify a BGP router in the BGP network, generally used for establishing BGP neighbor relationships, detecting routing loops, selecting optimal paths, and troubleshooting network issues.

5. Click **Create**.

## Configure an External IP Address Pool with L2Advertisement or BGPAdvertisement by using the CLI

```
# ippool-with-L2advertisement.yaml
kind: IPAddressPool
apiVersion: metallb.io/v1beta1
metadata:
  name: test-ippool
  namespace: metallb-system
spec:
  addresses:
    - 13.1.1.1/24
  avoidBuggyIPs: true
---
kind: L2Advertisement
apiVersion: metallb.io/v1beta1
metadata:
  name: test-ippool
  namespace: metallb-system
spec:
  ipAddressPools:
    - test-ippool ①
  nodeSelectors:
    - matchLabels: {}
      matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
            - 192.168.66.210
```

BGP mode

```
# ippool-with-bgpadvertisement.yaml
kind: IPAddressPool
apiVersion: metallb.io/v1beta1
metadata:
  name: test-pool-bgp
  namespace: metallb-system
spec:
  addresses:
    - 4.4.4.3/23
  avoidBuggyIPs: true
---
kind: BGPAdvertisement
apiVersion: metallb.io/v1beta1
metadata:
  name: test-pool-bgp
  namespace: metallb-system
spec:
  ipAddressPools:
    - test-pool-bgp
  nodeSelectors:
    - matchLabels:
        alertmanager: 'true'
  peers:
    - test-bgp-example
```

```
kubectl apply -f ippool-with-L2advertisement.yaml -f ippool-with-bgpadvertisement.yaml
```

## Troubleshooting MetalLB

Symptom	Possible Cause	Resolution
No external IP assigned	No valid IPAddressPool or pool misconfigured	Verify IP range and namespace
Pods CrashLoop	Speaker or Controller RBAC missing	Check Operator permissions

Symptom	Possible Cause	Resolution
BGP not established	ASN mismatch or peer unreachable	Check <code>BGPPeer</code> spec and network routes
L2 not working	Wrong VLAN or ARP filtering	Use <code>arping</code> to verify broadcast reachability

To see more [Troubleshooting MetalLB](#) ↗

# Configure GatewayAPI Gateway

---

## TOC

### [Prerequisites](#)

Configuration Via Web Console

Configuration Via YAML

Introduction

Service Type

LoadBalancer (**Recommended**)

NodePort

ClusterIP

Listener

Listener Support Kind

Allow Route NS

TLS

Hostname

Intersection Rule For Listener's Hostname And Route's Hostnames

Companion EnvoyProxy

Image Repository

Next Step

---

## Prerequisites

Please ensure that you have read the [Installation](#) documentation before proceeding.

## Configuration Via Web Console

1. Navigate to `Alauda Container Platform -> Networking -> Gateways`
2. Click on `Create Gateway` button
3. In `Create Gateway` page select `envoy-gateway-operator-cpaas-default` under GatewayClass, it will display the following configuration options:

Field	Description	YAML Path
Name	name	gateway: <code>.metadata.name</code> envoygateway: <code>.metadata.name</code>
GatewayClass	which GatewayClass it uses	gateway: <code>.spec.gatewayClassName</code>
Service Type	<a href="#">service type</a>	envoygateway: <code>.spec.provider.kubernetes.envoyService</code>
Service Annotation	service annotation	envoygateway: <code>.spec.provider.kubernetes.envoyService.annotation</code>
Resource Limits	deployment resource limits	envoygateway: <code>.spec.provider.kubernetes.envoyDeployment.contain</code>
Replicas	deployment replicas	envoygateway: <code>.spec.provider.kubernetes.envoyService</code>
Node Labels	deployment node labels	envoygateway: <code>.spec.provider.kubernetes.envoyService.nodeSelect</code>
Listener	<a href="#">listener</a>	gateway: <code>.spec.listeners</code>

**WARNING**

The Web Console form only supports GatewayClasses created by `EnvoyGatewayCt l`. For other GatewayClasses, use the YAML editor.

#### NOTE

When using an `EnvoyGatewayCt l`-created `GatewayClass`, the Web Console automatically creates a companion envoyproxy resource matching the Gateway's name and namespace.

## Configuration Via YAML

Empty content area for configuration details.



```

---
apiVersion: gateway.envoyproxy.io/v1alpha1
kind: EnvoyProxy
metadata:
  name: demo 8
  namespace: demo
spec:
  provider:
    kubernetes:
      envoyService:
        type: ClusterIP 9
      envoyDeployment:
        replicas: 1
        container:
          imageRepository: registry.alauda.cn:60080/acp/envoyproxy/envoy
10
        resources: 11
          limits:
            cpu: '1'
            memory: 1Gi
          requests:
            cpu: '1'
            memory: 1Gi
        type: Kubernetes 12

```

- 1 Companion envoyproxy
- 2 Specify which `GatewayClass` it belongs to
- 3 Listener
- 4 Listener hostname
- 5 Listener support kind
- 6 Listener allow ns
- 7 TLS configuration
- 8 Name of companion envoyproxy
- 9 Service type
- 10 Please keep and do not modify the default value of `repository`
- 11 Resource for `envoy-proxy instance`
- 12 Please keep and do not modify it

# Introduction

## Service Type

Service Type essentially configures how the gateway is exposed. There are three modes: LoadBalancer NodePort and ClusterIP.

### LoadBalancer (Recommended)

The advantage is ease of use, and high-availability load balancing capabilities. To use LoadBalancer, the cluster must have LoadBalancer support, which can be enabled via [MetalLB](#).

### NodePort

The advantage is that it doesn't require any external dependencies.

However, using NodePort has these disadvantages:

- Can only be used in clusters with fewer than 16 nodes, otherwise the gateway status may become abnormal.
- When using NodePort, Kubernetes assigns NodePort port numbers that differ from the service's own ports. You must use the NodePort port number for access, not the service port.
- The service can be accessed via any node IP address in the cluster, which may pose potential security risks

### How to Get The Correct Port When Using NodePort

```
kubectl get svc -n ${ENVOYGATEWAYCTL_NS} -l gateway.envoyproxy.io/owning-gateway-name=${GATEWAY_NAME} -o=jsonpath='{.items[0].spec.ports[?(@.port=${PORT})].nodePort}'
```

The output is the NodePort

### ClusterIP

Very convenient if you don't need external exposure.

## Listener

Listener defines the port and protocol for the gateway to listen on. In HTTP or HTTPS protocols, different hostnames can be treated as a different listener.

You cannot create a listener with a conflicting port or protocol or hostname.

You must create at least one listener in the `Gateway`.

## Listener Support Kind

Each listener supports different route kinds based on its protocol:

Listener Protocol	Supported Route Kind
HTTP	HTTPRoute
HTTPS	HTTPRoute, GRPCRoute
TLS	TLSRoute
TCP	TCPRoute
UDP	UDPRoute

When configuring routes, ensure they match the protocol of the listener they'll attach to. For example, you cannot attach an HTTPRoute to a TCP listener.

## Allow Route NS

By default, Routes can only attach to a Gateway in the same namespace. To allow cross-namespace routing, you need to specify which namespaces are allowed to attach Routes to this Gateway's listener using the `allowedRoutes` field.

For more information, please reference [attach to gateway create on other ns](#).

## TLS

By default, you can only use `secret` created in the same namespace. Otherwise, please refer to [use secret create on other ns](#).

By default, we use `Terminate` mode. Otherwise please refer to [ssl passthrough](#).

## Hostname

The hostname in a listener is a unique identifier for listeners that have the same protocol. you cannot add or update a conflicting listener in a gateway.

## Intersection Rule For Listener's Hostname And Route's Hostnames

When a request arrives, it is matched against the **intersection** of the Listener's hostname and the Route's hostnames. Only hostnames in the intersection are used for routing traffic.

Listener Hostname	Route Hostnames	Intersection Result	Example
No hostname	No hostnames	Matches all hosts	Any incoming host header is accepted
No hostname	Has hostnames (e.g., <code>api.example.com</code> )	All Route hostnames	Only requests with <code>api.example.com</code> are matched
Has hostname (e.g., <code>api.example.com</code> )	No hostnames	All Listener hostnames	Only requests with <code>api.example.com</code> are matched
Has hostname (e.g., <code>api.example.com</code> )	Has matching exact hostname (e.g., <code>api.example.com</code> )	Exact match hostname	Only requests with <code>api.example.com</code> are matched
Has wildcard (e.g., <code>*.example.com</code> )	Has matching hostnames (e.g., <code>api.example.com</code> , <code>web.example.com</code> )	Matching specific hostnames	Requests with <code>api.example.com</code> or <code>web.example.com</code> are matched

Listener Hostname	Route Hostnames	Intersection Result	Example
Has hostname (e.g., <code>api.example.com</code> )	Has non-matching hostnames (e.g., <code>web.example.com</code> )	<b>No intersection</b> - Route status is abnormal	Route cannot process traffic

**NOTE**

Wildcards ( `*` ) perform suffix matching. For example, `*.example.com` matches `foo.example.com` and `bar.example.com`, but not `example.com`.

**WARNING**

No intersection - Route status is abnormal, and cannot process traffic.

## Companion EnvoyProxy

`Envoy Gateway` provides different granularities for controlling gateway deployments. We recommend creating a dedicated `EnvoyProxy` resource for each Gateway and referencing it through the Gateway's `.spec.infrastructure.parametersRef` field.

This one-to-one mapping approach provides better isolation and more granular control over deployment configurations such as replicas, resources, and scheduling constraints.

For other deployment configuration methods, please refer to [deployment-mode](#).

## Image Repository

This is the default value. It will be replaced by the actual image repository of the current cluster, please do not modify it.

# Next Step

[Configure GatewayAPI Route](#)

# Configure GatewayAPI Route

---

## TOC

### Prerequisites

Configuration Via Web Console

- Create HTTPRoute

- Create TCP/UDP Route

Configuration Via YAML

Introduction

- Hostnames

- Publish to Listener

Rules

- Matches

- Filters

- Backend

- Advanced HTTPRoute Rule Settings

Next Step

Related Tasks

---

## Prerequisites

Please ensure that you have read the [Installation](#) documentation before proceeding.

---

# Configuration Via Web Console

1. Navigate to `Alauda Container Platform -> Networking -> Routes`
2. Click on `Create Route` button

## Create HTTPRoute

Field	Description	YAML Path
Publish to Listener	<a href="#">publish to listener</a>	<code>.spec.parentRefs</code>
Hostnames	<a href="#">hostnames</a>	<code>.spec.hostnames</code>
Matches	<a href="#">matches</a>	<code>.spec.rules[].matches</code>
Filters	<a href="#">filters</a>	<code>.spec.rules[].filters</code>
Backend Instance	<a href="#">backend</a>	<code>.spec.rules[].backendRefs</code>
Session Affinity	<a href="#">session affinity</a>	<code>.spec.rules[].sessionPersistence</code>

## Create TCP/UDP Route

Field	Description	YAML Path
Publish to Listener	<a href="#">publish to listener</a>	<code>.spec.parentRefs</code>
Backend Instance	<a href="#">backend</a>	<code>.spec.rules[].backendRefs</code>

## Configuration Via YAML

Empty content area for configuration details.

```
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: demo-443
  namespace: demo
spec:
  hostnames: ①
    - example.com
  parentRefs: ②
    - group: gateway.networking.k8s.io
      kind: Gateway
      name: demo
      sectionName: https
  rules: ③
    - backendRefs: ④
        - group: ''
          kind: Service
          name: echo-resty
          namespace: demo-space
          port: 80
          weight: 100
      filters: [] ⑤
      matches: ⑥
        - path:
            type: Exact
            value: /a
      sessionPersistence: ⑦
        type: Cookie
        sessionName: a
    ---
apiVersion: gateway.networking.k8s.io/v1alpha2
kind: TCPRoute
metadata:
  name: tcp
  namespace: demo-space
spec:
  parentRefs:
    - group: gateway.networking.k8s.io
      kind: Gateway
      name: demo
      sectionName: tcp
  rules:
    - backendRefs:
```

```
- group: ''
  kind: Service
  name: echo-resty
  port: 80
  weight: 100

---
apiVersion: gateway.networking.k8s.io/v1alpha2
kind: UDPRoute
metadata:
  name: udp
  namespace: demo
spec:
  parentRefs:
    - group: gateway.networking.k8s.io
      kind: Gateway
      name: demo
      namespace: demo
      sectionName: udp
  rules:
    - backendRefs:
        - group: ''
          kind: Service
          name: echo-resty
          namespace: demo
          port: 53
          weight: 100
```

- 1 Hostnames
- 2 Publish to listener
- 3 Rules
- 4 Backend
- 5 Filters
- 6 Matches
- 7 Session

## Introduction

Each route is a CR defined by the `GatewayAPI` specification. For detailed information about the fields and configuration options for each route type, please refer to the official documentation: - [HTTPRoute specification ↗](#) - [TCPRoute specification ↗](#) - [UDPRoute specification ↗](#) - [GRPCRoute specification ↗](#) - [TLSRoute specification ↗](#)

## Hostnames

The `hostnames` field in a Route is a string array. It follows the [Hostname Intersection Rules](#).

## Publish to Listener

- The `sectionName` is the listener name.
- Routes can only be attached to [listeners that support their specific kind](#).
- By default, routes can only be attached to listeners where the `Gateway` is in the same namespace. For cross-namespace attachment, please refer to [attaching to a gateway created in another namespace](#).

## Rules

Each route can contain multiple rules. Each rule consists of the following components:

### Matches

Defines the conditions that must be met for a request to be routed by this rule.

A rule can have multiple matches:

- Each match consists of multiple conditions (e.g., path, headers, query parameters, method)
- Conditions **within** a match use **AND** logic (all must be satisfied)
- Matches **between** each other use **OR** logic (any match can satisfy the rule)

**Example:** If Match-1 requires `path=/api` AND `header=v1`, and Match-2 requires `query=test`, then a request is routed if it matches either `(path=/api AND header=v1)` OR `(query=test)`.

### Match Condition Types

Object	Method	Value Types	Description	Value Requirements
Path				
	Exact	path (string)	<p>Matches the URL path exactly and with case sensitivity. This means that an exact path match on /abc will only match requests to /abc, <b>NOT</b> /abc/, /Abc, or /abcd.</p>	Must start with /, no consecutive //.
	PathPrefix	path (string)	<p>Matches based on a URL path prefix split by /. Matching is case-sensitive and done on a <b>path element by element basis</b>. For example, the paths /abc, /abc/, and /abc/def would all match the prefix /abc, but the path /abcd would not.</p>	Must start with /, no consecutive //.

Object	Method	Value Types	Description	Value Requirements
	RegularExpression	path (string)	Regex Engine: RE2.	for example, /api/v1/.*
<b>Header</b>				
	Exact	name (header key) + value	Exact header value match.	
	RegularExpression	name (header key) + value	Regex Engine: RE2.	
<b>QueryParam</b>				
	Exact	name (param key) + value	Exact query parameter value match.	Parameter value: 1-1024 characters
	RegularExpression	name (param key) + value	Regex Engine: RE2.	
<b>Method</b>	-	method name	HTTP method match.	GET , HEAD , POST , PUT , DELETE , CONNECT , OPTIONS , TRACE , PATCH

### Match Condition References

Condition Type	Official Documentation
Path	<a href="#">HTTPPathMatch ↗</a>
Headers	<a href="#">HTTPHeaderMatch ↗</a>
QueryParams	<a href="#">HTTPQueryParamMatch ↗</a>
Method	<a href="#">HTTPMethod ↗</a>

## Filters

Specifies transformations or modifications to apply to requests/responses.

### Filter Types

Type	Method	Value Types	Description
<b>RequestHeaderModifier</b>	Set	name (string) + value (string)	Overwrites request header with given name and value
	Add	name (string) + value (string)	Adds header to request, appending to existing values
	Remove	[]string	Removes specified headers from request (case-insensitive)

Type	Method	Value Types	Description
<b>ResponseHeaderModifier</b>	Set	name (string) + value (string)	Overwrites response header with given name and value
	Add	name (string) + value (string)	Adds header to response, appending to existing values
	Remove	[]string	Removes specified headers from response (case-insensitive)
<b>RequestRedirect</b>	Scheme	string	Scheme for Location header (http/https)
	Hostname	PreciseHostname	Hostname for Location header
	ReplaceFullPath	string	Replace entire request path

Type	Method	Value Types	Description
	ReplacePrefixMatch	string	Replace matched path prefix
	Port	PortNumber	Port for Location header
	StatusCode	int	HTTP redirect status code
URLRewrite	Hostname	PreciseHostname	Hostname to rewrite in request
	ReplaceFullPath	string	Replace entire request path
	ReplacePrefixMatch	string	Replace matched path prefix
CORS	AllowOrigins	[]string	List of allowed origins for CORS requests
	AllowMethods	[]HTTPMethod	List of allowed

Type	Method	Value Types	Description
			HTTP methods
	<code>AllowHeaders</code>	<code>[]string</code>	List of allowed headers in CORS requests
	<code>ExposeHeaders</code>	<code>[]string</code>	List of headers exposed to client in response
	<code>MaxAge</code>	Duration	Cache duration for CORS preflight response
	<code>AllowCredentials</code>	bool	Whether credentials are allowed in CORS requests

**Notes:**

- `RequestRedirect` and `URLRewrite` cannot be used together on the same rule
- `ReplacePrefixMatch` is only compatible with a `PathPrefix` `HTTPRouteMatch`
- Header names are case-insensitive per RFC 7230
- Multiple values for same header must use RFC 7230 comma-separated format

**Filter References**

Filter Type	Official Documentation
RequestHeaderModifier	<a href="#">HTTPHeaderFilter ↗</a>
ResponseHeaderModifier	<a href="#">HTTPHeaderFilter ↗</a>
RequestRedirect	<a href="#">HTTPRequestRedirectFilter ↗</a>
URLRewrite	<a href="#">HTTPURLRewriteFilter ↗</a>
CORS	<a href="#">HTTPCORSFilter ↗</a>
RequestMirror	<a href="#">HTTPRequestMirrorFilter ↗</a>
HTTPExternalAuthFilter	<a href="#">HTTPExternalAuthFilter ↗</a>

## Backend

Defines the target service(s) where matching requests should be forwarded.

Each service can have a `weight` field to specify the proportion of traffic to be routed to that service.

## Advanced HTTPRoute Rule Settings

HTTPRoute rules support additional configuration fields, such as retry policies, timeouts, and other traffic management parameters.

### Timeouts

Field	Specification
<code>.spec.rules[].timeouts</code>	<a href="#">HTTPRouteTimeouts ↗</a>

### Retry

Field	Specification
<code>.spec.rules[].retry</code>	<a href="#">HTTPRouteRetry ↗</a>

## Session Persistence

Configures session affinity settings to ensure requests from the same client are routed to the same backend.

Field	Specification
<code>.spec.rules[].sessionPersistence</code>	<a href="#">SessionPersistence</a> ↗

## Next Step

[Tasks for Envoy Gateway](#)

## Related Tasks

- [How to attach to listener created in other namespace](#)

# Configure ALB

## WARNING

ALB has been deprecated. Please use the [ingress-nginx-operator](#) or [envoy-gateway](#) instead.

## TOC

### ALB

- Prerequisites

### Configure ALB

- Resource-Related Configuration

- Networking Configuration

- project configuration

- tweak configuration

- Operation On ALB

- Creating

- Update

- Delete

- Frontend

- Prerequisites

- Configure Frontend

- Operation On Frontend

- Creating

- Subsequent Actions

- Related Operations

## Rule

### Prerequisites

match request with dslx and priority

dslx

priority

### Action

### Backend

backend protocol

Service Group and Session Affinity Policy

### Operation On Rule

Using web console

using the CLI

## Https

Certificate Annotation in Ingress

Certificate in Rule

TLS Mode

## Ingress

Ingress sync

## Logs and Monitoring

Viewing Logs

Monitoring Metrics

---

# ALB

ALB is a custom resource that represents a load balancer. The alb-operator, which is embedded by default in all clusters, watches for create/update/delete operations on ALB resources and creates corresponding deployments and services in response.

For each ALB, a corresponding Deployment watches all Frontends and Rules attached to that ALB and routes requests to backends based on those configurations.

---

## Prerequisites

The high availability of the **Load Balancer** requires a VIP. Please refer to [Configure VIP](#).

## Configure ALB

There are three parts to an ALB configuration.

```
# test-alb.yaml
apiVersion: crd.alauda.io/v2beta1
kind: ALB2
metadata:
  name: alb-demo
  namespace: cpaas-system
spec:
  address: 192.168.66.215
  config:
    vip:
      enableLbSvc: false
      lbSvcAnnotations: {}
  networkMode: host
  nodeSelector:
    cpu-model.node.kubevirt.io/Nehalem: 'true'
  replicas: 1
  resources:
    alb:
      limits:
        cpu: 200m
        memory: 256Mi
      requests:
        cpu: 200m
        memory: 256Mi
    limits:
      cpu: 200m
      memory: 256Mi
    requests:
      cpu: 200m
      memory: 256Mi
  projects:
    - ALL_ALL
  type: nginx
```

## Resource-Related Configuration

resource related field describes the deployment configuration for the alb.

Field	Type	Description
<code>.spec.config.nodeSelector</code>	map[string]string	the node selector for the alb
<code>.spec.config.replicas</code>	int,optional default 3	the number of replicas for the alb
<code>.spec.config.resources.limits</code>	k8s container-resource,optional	limit of nginx container of alb
<code>.spec.config.resources.requests</code>	k8s container-resource,optional	request of nginx container of alb
<code>.spec.config.resources.alb.limits</code>	k8s container-resource,optional	limit of alb container of alb
<code>.spec.config.resources.alb.requests</code>	k8s container-resource,optional	request of alb container of alb
<code>.spec.config.antiAffinityKey</code>	string,optional default local	k8s antiAffinityKey

## Networking Configuration

Networking fields describe how to access the ALB. For example, in `host` mode, alb will use hostnetwork, and you can access the ALB via the node IP.

Field	Type	Description
<code>.spec.config.networkMode</code>	string: <code>host</code> or <code>container</code> , optional, default <code>host</code>	In <code>container</code> mode, the operator creates a LoadBalancer Service and uses its address as the ALB address.
<code>.spec.address</code>	string,required	you could manually specify the address of alb

Field	Type	Description
<code>.spec.config.vip.enableLbSvc</code>	bool, optional	Automatically true in <code>container</code> mode.
<code>.spec.config.vip.lbSvcAnnotations</code>	map[string]string, optional	Extra annotations for the LoadBalancer Service.

## project configuration

Field	Type
<code>.spec.config.projects</code>	[]string,required
<code>.spec.config.portProjects</code>	string,optional
<code>.spec.config.enablePortProject</code>	bool,optional

Adding an ALB to a project means:

1. In the web UI, only users in the given project can find and configure this ALB.
2. This ALB will handle ingress resources belonging to this project. Please refer to [ingress-sync](#).
3. In the web UI, rules created in project X cannot be found or configured under project Y.

If you enable port project and assign a port range to a project, this means:

1. You cannot create ports that do not belong to the port range assigned to the project.

## tweak configuration

there are some global config which can be tweaked in alb cr.

- [Bind NIC](#)
- [Ingress sync](#)

## Operation On ALB

# Creating

## Using the web console.

The screenshot shows the 'Create Load Balancers' configuration page in the Alauda web console. The page is organized into several sections:

- Name:** Includes a required 'Name' field (with a hint: 'Starts with a letter. Ends with a letter or number. Contains only lower case letters, numbers, and -') and a 'Display Name' field.
- Network Configuration:**
  - Network Mode:** 'Host network' is selected, with 'Container network' as an alternative.
  - Service:** A toggle switch is currently turned off. A note below states: 'When enabled, a LoadBalancer type service will be created, providing an access address for the load balancer through services. When disabled, refer to the help to configure the access address for the load balancer.'
  - Access Address:** A text input field with a placeholder 'Enter a domain name or IPv4/IPv6 address' and an 'Add' button below it.
- Resource Configuration:**
  - Specification:** 'Small scale' is selected, with options for 'Medium scale', 'Large scale', and 'Custom'. Descriptions are provided for each scale.
  - Resource Limits:** 'CPU' is set to 200 and 'Memory' is set to 256 Mi.
  - Deployment type:** 'Standalone' is selected, with 'High availability' as an alternative.
  - Replicas:** A numeric input field is set to 1.
  - Node Labels:** A dropdown menu for selecting node labels.
  - Allocated By:** 'Instance' is selected, with 'Port' as an alternative.
  - Allocated Projects:** 'All Projects' is selected, with 'Specific projects' and 'None' as alternatives.

Some common configuration is exposed in the web UI. Follow these steps to create a load balancer:

1. Navigate to **Administrator**.
2. In the left sidebar, click on **Network Management > Load Balancer**.
3. Click on **Create Load Balancer**.

Each input item in the web UI corresponds to a field of the CR:

Parameter	Description
Assigned Address	<code>.spec.address</code>
Allocated By	<code>Instance</code> means project mode, and you could select project below, port means port-project mode, you could assign port-range after create alb

## Using the CLI.

```
kubectl apply -f test-alb.yaml -n cpaas-system
```

## Update

### Using the web console

#### NOTE

Updating the load balancer will cause a service interruption for 3 to 5 minutes. Please choose an appropriate time for this operation!

1. Enter **Administrator**.
2. In the left navigation bar, click **Network Management > Load Balancer**.
3. Click **> Update**.
4. Update the network and resource configuration as needed.
  - Please set specifications reasonably according to business needs. You can also refer to the relevant [How to properly allocate CPU and memory resources](#) for guidance.
  - **Internal routing** only supports updating from **Disabled** state to **Enabled** state.
5. Click **Update**.

## Delete

### Using the web console

#### NOTE

After deleting the load balancer, the associated ports and rules will also be deleted and cannot be restored.

1. Enter **Administrator**.
2. In the left navigation bar, click **Network Management > Load Balancer**.
3. Click **> Delete**, and confirm.

## Using the CLI

```
kubectl delete alb2 alb-demo -n cpaas-system
```

## Frontend

Frontend is a custom resource that defines the listener port and protocol for an ALB.

Supported protocols: L7 (http|https|grpc|grpcs) and L4 (tcp|udp).

- In L4 Proxy use frontend to configure backend service directly.
- In L7 Proxy use frontend to configure listener ports, and use [rule](#) to configure backend service.

If you need to add an HTTPS listener port, you should also contact the administrator to assign a TLS certificate to the current project for encryption.

## Prerequisites

Create a ALB first.

## Configure Frontend

```
# alb-frontend-demo.yaml
apiVersion: crd.alauda.io/v1
kind: Frontend
metadata:
  labels:
    alb2.cpaas.io/name: alb-demo ①
  name: alb-demo-00080 ②
  namespace: cpaas-system
spec:
  port: 80 ③
  protocol: http ④
  certificate_name: '' ⑤
  backendProtocol: 'http' ⑥
  serviceGroup: ⑦
    session_affinity_policy: '' ⑧
  services:
    - name: hello-world
      namespace: default
      port: 80
      weight: 100
```

- ① alb label: Required, indicate the ALB instance to which this `Frontend` belongs to.
- ② frontend name: Format as `$alb_name-$port`.
- ③ port: which port which listen on.
- ④ protocol: what protocol this port uses.
  - L7 protocol https|http|grpcs|grpc and L4 protocol tcp|udp.
  - When selecting HTTPS, a certificate must be added; adding a certificate is optional for the gRPC protocol.
  - When selecting the gRPC protocol, the backend protocol defaults to gRPC, which does not support session persistence. If a certificate is set for the gRPC protocol, the load balancer will unload the gRPC certificate and forward the unencrypted gRPC traffic to the backend service.
  - If using a Google GKE cluster, a load balancer of the same **container network type** cannot have both TCP and UDP listener protocols simultaneously.
- ⑤ certificate\_name: for grpcs and https protocol which the default cert used, Format as `$secret_ns/$secret_name`.

6 backendProtocol: what protocol the backend service uses.

7 Default `serviceGroup` :

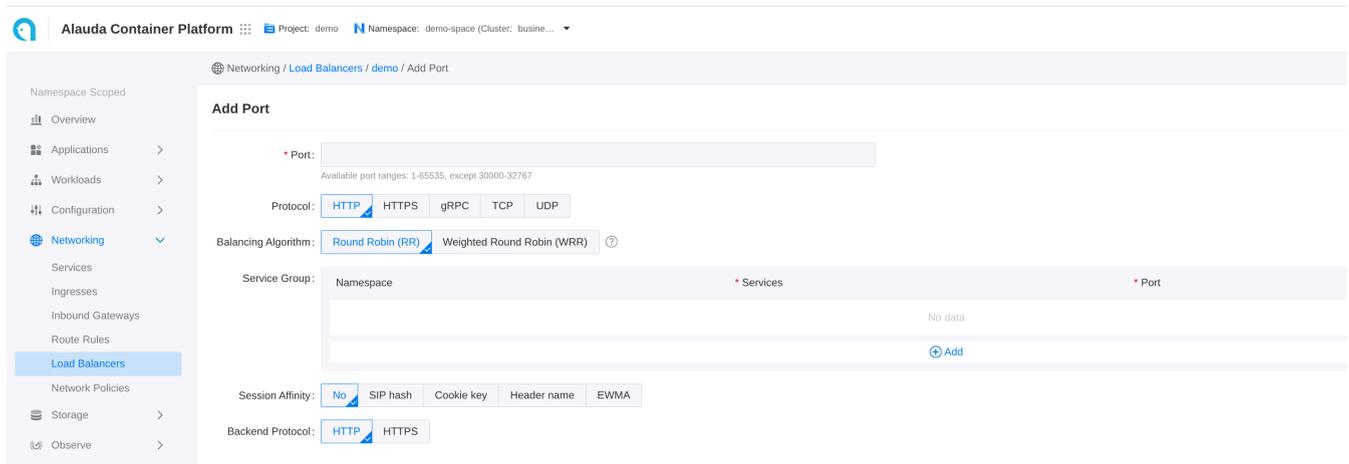
- L4 proxy: required. ALB forwards traffic to the default service group directly.
- L7 proxy: optional. ALB first matches Rules on this Frontend; if none match, it falls back to the default `serviceGroup` .

8 `session_affinity_policy`

## Operation On Frontend

### Creating

#### using the web console



1. Go to **Container Platform**.
2. In the left navigation bar, click **Network** > **Load Balancing**.
3. Click the name of the load balancer to enter the details page.
4. Click **Add Port**.

Each input item on the web UI corresponds to a field of the CR

Parameter	Description
Session Affinity	<code>.spec.serviceGroup.session_affinity_policy</code>

### using the CLI

```
kubectl apply -f alb-frontend-demo.yaml -n cpaas-system
```

## Subsequent Actions

For traffic from HTTP, gRPC, and HTTPS ports, in addition to the default internal routing group, you can set more varied back-end service matching [rules](#). The load balancer will initially match the corresponding backend service according to the set rules; if the rule match fails, it will then match the backend services corresponding to the aforementioned internal routing group.

## Related Operations

You can click the  icon on the right side of the list page or click **Actions** in the upper right corner of the details page to update the default route or delete the listener port as needed.

### NOTE

If the resource allocation method of the load balancer is **Port**, only administrators can delete the related listener ports in the **Administrator** view.

## Rule

Rule is a Custom Resource(CR) that defines how incoming requests are matched and processed by the ALB.

Ingresses handled by ALB can be [auto-translated to rules](#).

## Prerequisites

- [Configure ALB](#)
- [Configure Frontend](#)

Here is a demo rule to give you a quick first impression of how to use rules.

**NOTE**

rule must be attached to a frontend and alb via label.

```
apiVersion: crd.alauda.io/v1
kind: Rule
metadata:
  labels:
    alb2.cpaas.io/frontend: alb-demo-00080 1
    alb2.cpaas.io/name: alb-demo 2
  name: alb-demo-00080-test
  namespace: cpaas-system
spec:
  backendProtocol: 'https' 3
  certificate_name: 'a/b' 4
  dslx: 5
  - type: URL
    values:
      - - STARTS_WITH
        - /
  priority: 4 6
  serviceGroup: 7
  services:
    - name: hello-world
      namespace: default
      port: 80
      weight: 100
```

- 1 Required, indicate the `Frontend` to which this rule belongs.
- 2 Required, indicate the ALB to which this rule belongs.
- 3 `backendProtocol`
- 4 `certificate_name`
- 5 `dslx`
- 6 The lower the number, the higher the priority.
- 7 `serviceGroup`

# match request with dslx and priority

## dslx

DSLX is a domain-specific language used to describe the matching criteria. For example, the rule below matches a request that satisfies **all** the following criteria:

- url starts with /app-a **or** /app-b
- method is post
- url param's group is vip
- host is \*.app.com
- header's location is east-1 or east-2
- has a cookie name is uid
- source IPs come from 1.1.1.1-1.1.1.100

```
ds1x:
  - type: METHOD
    values:
      - EQ
      - POST
  - type: URL
    values:
      - STARTS_WITH
      - /app-a
      - STARTS_WITH
      - /app-b
  - type: PARAM
    key: group
    values:
      - EQ
      - vip
  - type: HOST
    values:
      - ENDS_WITH
      - .app.com
  - type: HEADER
    key: LOCATION
    values:
      - IN
      - east-1
      - east-2
  - type: COOKIE
    key: uid
    values:
      - EXIST
  - type: SRC_IP
    values:
      - RANGE
      - '1.1.1.1'
      - '1.1.1.100'
```

## priority

Priority is an integer ranging from 0 to 10, where lower values indicate higher priority. To configure the priority of a rule in ingress, you can use the following annotation format:

```
# alb.cpaas.io/ingress-rule-priority-$rule_index-$path_index
alb.cpaas.io/ingress-rule-priority-0-0: '10'
```

For rules, simply set the priority directly in `.spec.priority` using an integer value.

## Action

After a request matches a rule, you can apply the following actions to the request

Feature	Description	Link
Timeout	Configures the timeout settings for requests.	<a href="#">timeout</a>
Redirect	Redirects incoming requests to a specified URL.	<a href="#">redirect</a>
CORS	Enables Cross-Origin Resource Sharing (CORS) for the application.	<a href="#">cors</a>
Header Modification	Allows modification of request or response headers.	<a href="#">header modification</a>
URL Rewrite	Rewrites the URL of incoming requests before forwarding them.	<a href="#">url-rewrite</a>
WAF	Integrates Web Application Firewall (WAF) for enhanced security.	<a href="#">waf</a>
OTEL	Enables OpenTelemetry (OTEL) for distributed tracing and monitoring.	<a href="#">otel</a>
Keepalive	Enables or disables the keepalive feature for the application.	<a href="#">keepalive</a>

## Backend

### backend protocol

By default, the backend protocol is set to HTTP. If you want to use TLS re-encryption, you can configure it as HTTPS.

## Service Group and Session Affinity Policy

You can configure one or more services within a rule.

By default, the ALB uses a round-robin (RR) algorithm to distribute requests among backend services. However, you can assign weights to individual services or choose a different load-balancing algorithm.

For more details, refer to [Balance Algorithm](#).

## Operation On Rule

### Using web console

The screenshot shows the 'Add Rule' configuration page in the Alauda Container Platform web console. The page is titled 'Networking / Load Balancers / demo / demo-08888 / Add Rule'. The left navigation menu is expanded to 'Load Balancers'. The main configuration area includes the following fields and options:

- Load Balancers:** demo
- Port:** 8888
- Protocol:** HTTP
- Description:** Please input description
- Balancing Algorithm:** Round Robin (RR) (selected), Weighted Round Robin (WRR)
- Service Group:** Namespace: demo-space, Services: (empty), Port: (empty)
- Rule:** Type: Domains (selected), Parameters: URL, IP, Headers, Cookie, URL Param
- Session Affinity:** No (selected), SIP hash, Cookie key, Header name, EWMA
- URL Rewrite:** (off)
- Backend Protocol:** HTTP (selected), HTTPS
- URL Redirect:** (off)
- Priority:** 5

At the bottom of the page, there is a note: "Set the priority of the rule selected by the traffic. Numbers 1-10 are supported. The smaller the value, the priority will be selected. That is, when the traffic is matched by multiple rules, only the rule with the smallest priority value is selected and the rule is applied."

1. Go to **Container Platform**.
2. Click on **Network > Load Balancing** in the left navigation bar.
3. Click on the name of the load balancer.
4. Select the listener port name.
5. Click **Add Rule**.
6. Refer to the following descriptions to configure the relevant parameters.

## 7. Click **Add**.

Each input item on the webui corresponds to a field of the CR

## using the CLI

```
kubectl apply -f alb-rule-demo.yaml -n cpaas-system
```

## Https

If the frontend protocol (ft) is HTTPS or GRPCS, the rule can also be configured to use HTTPS.

You can specify the certificate either in the rule or in the ingress to match the certificate for that specific port.

Termination is supported, and re-encryption is possible if the backend protocol is HTTPS. However, you **cannot** specify a certificate for communication with the backend service.

## Certificate Annotation in Ingress

Certificates can be referenced across namespaces via annotation.

```
alb.networking.cpaas.io/tls: qq.com=cpaas-system/dex.tls,qq1.com=cpaas-system/dex1.tls
```

## Certificate in Rule

In `.spec.certificate_name`, the format is `$secret_namespace/$secret_name`

## TLS Mode

### Edge Mode

In edge mode, the client communicates with the ALB using HTTPS, and ALB communicates with backend services using HTTP protocol. To achieve this:

1. create ft use https protocol
2. create rule with backend protocol http, and specify cert via `.spec.certificate_name`

## Re-encrypt Mode

In re-encrypt mode, the client communicates with the ALB using HTTPS, and ALB communicates with backend services using HTTPS protocol. To achieve this:

1. create ft use https protocol
2. create rule with backend protocol https, and specify cert via `.spec.certificate_name`

## Ingress

### Ingress sync

Each ALB creates an IngressClass with the same name and handles ingresses within the same project.

When an ingress namespace has a label like `cpaas.io/project: demo`, it indicates that the ingress belongs to the `demo` project.

ALBs that have the project name `demo` in their `.spec.config.projects` configuration will automatically translate these ingresses into rules.

#### NOTE

ALB listens to ingress and automatically creates a `Frontend` or `Rule`. `source` field is defined as follows:

1. `spec.source.type` currently only supports `ingress`.
2. `spec.source.name` is ingress name.
3. `spec.source.namespace` is ingress namespace.

## SSL strategy

For ingresses that do not have certificates configured, ALB provides a strategy to use a default certificate.

You can configure the ALB custom resource with the following settings:

- `.spec.config.defaultSSLStrategy`: Defines the SSL strategy for ingresses without certificates
- `.spec.config.defaultSSLCert`: Sets the default certificate in the format `$secret_ns/$secret_name`

Available SSL strategies:

- **Never**: Do not create rules on HTTPS ports (default behavior)
- **Always**: Create rules on HTTPS ports using the default certificate

## Logs and Monitoring

By combining logs and monitoring data, you can quickly identify and resolve load balancer issues.

### Viewing Logs

1. Go to **Administrator**.
2. In the left navigation bar, click on **Network Management > Load Balancer**.
3. Click on ***Load Balancer Name***.
4. In the **Logs** tab, view the logs of the load balancer's runtime from the container's perspective.

### Monitoring Metrics

#### NOTE

The cluster where the load balancer is located must deploy monitoring services.

1. Go to **Administrator**.
2. In the left navigation bar, click on **Network Management > Load Balancer**.
3. Click on ***Load Balancer Name***.
4. In the **Monitoring** tab, view the metric trend information of the load balancer from the node's perspective.
  - **Usage Rate**: The real-time usage of CPU and memory by the load balancer on the current node.
  - **Throughput**: The overall incoming and outgoing traffic of the load balancer instance.

For more detailed information about monitoring metrics please refer to [ALB Monitoring](#).

# Configure NodeLocal DNSCache

---

## TOC

### [Overview](#)

[Key Features](#)

[Important Notes](#)

[Installation](#)

[Install via Marketplace](#)

[How It Works](#)

[Architecture](#)

[Configuration](#)

[Network Policy Configuration](#)

---

## Overview

NodeLocal DNSCache is a cluster plugin that improves cluster DNS performance by running a DNS caching proxy on cluster nodes. This plugin reduces DNS query latency and improves cluster stability by caching DNS responses locally on each node, minimizing the load on the central DNS service.

## Key Features

---

- **Local DNS Caching:** Caches DNS responses locally on each node to reduce query latency
- **Improved Performance:** Significantly reduces DNS lookup times for applications

## Important Notes

### WARNING

#### Deployment Considerations:

1. **Kube-OVN Underlay Mode:** The plugin does not support deployment in Kube-OVN Underlay mode. If deployed, it may cause DNS query failures.
2. **Kubelet Restart:** Deploying this plugin will cause the kubelet to restart.
3. **Pod Restart Required:** After the plugin is successfully deployed, it will not affect running Pods, but will only take effect on newly created Pods. When the CNI is Kube-OVN, you need to manually add the parameter "--node-local-dns-ip=(IP address of the local DNS cache server)" to the kube-ovn-controller.
4. **NetworkPolicy Configuration:** If NetworkPolicy is configured in the cluster, you need to additionally allow both from and to directions for the node CIDR and nodeLocalDNSIP in the networkPolicy to ensure proper communication.

### WARNING

#### 4.2.x Upgrade Notes

When upgrading this plugin from versions below 4.2.0 (excluding 4.2.0 itself) to 4.2.x, the following steps are required due to ResourcePatch compatibility issues:

#### Before Upgrade:

- Record the `--node-local-dns-ip` parameter value from the kube-ovn-controller ResourcePatch configuration
- Delete the ResourcePatch for the `deploy/kube-ovn-controller` resource

#### After Upgrade:

- Manually add the recorded `--node-local-dns-ip` parameter back to the kube-ovn-controller configuration

**Note:** This compatibility issue has been resolved in version 4.3 and above, so manual intervention is not required for upgrades to 4.3+.

## Installation

### Install via Marketplace

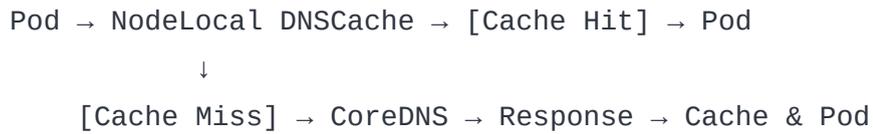
- Navigate to **Administrator > Marketplace > Cluster Plugins**.
- Search for "**Alauda Build of NodeLocal DNSCache**" in the plugin list.
- Click **Install** to open the installation configuration page.
- Configure the required parameters:

Parameter	Description	Example Value
<b>IP</b>	The IP address of the node local DNS cache server. For IPv4, it is recommended to use an address within the 169.254.0.0/16 range, preferably 169.254.20.10. For IPv6, it is recommended to use an address within the fd00::/8 range, preferably fd00::10.	169.254.20.10

- Review the deployment notes and ensure your environment meets the requirements.
- Click **Install** to complete the installation.
- Wait for the plugin status to change to "**Ready**".

## How It Works

### Architecture



## Configuration

### Network Policy Configuration

**Important:** If your cluster has NetworkPolicy enabled, you must configure proper rules to allow DNS traffic to the NodeLocal DNSCache. Without these rules, pods may not be able to resolve DNS queries.

When using NetworkPolicy, ensure the following DNS traffic is allowed:

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-dns-cache
spec:
  podSelector: {}
  policyTypes:
    - Ingress
    - Egress
  ingress:
    - from:
        - ipBlock:
            cidr: 169.254.20.10/32 # NodeLocal DNS IP address
      ports:
        - protocol: UDP
          port: 53
        - protocol: TCP
          port: 53
  egress:
    - to:
        - ipBlock:
            cidr: 169.254.20.10/32 # NodeLocal DNS IP address
      ports:
        - protocol: UDP
          port: 53
        - protocol: TCP
          port: 53
```

# Configure CoreDNS

---

## TOC

### Overview

#### Configuration

##### Host Alias

##### Node Selectors

##### Node Tolerations

---

## Overview

CoreDNS is the default DNS service for Kubernetes clusters. This guide describes how to configure host aliases, node selectors, and tolerations for CoreDNS.

## Configuration

1. Navigate to **Administrator > Marketplace > Cluster Plugins**.
2. Search for "**Alauda Build of CoreDNS**" and click **Update**.
3. Configure the following parameters:

### Host Alias

Configure custom DNS resolution entries.

---

Parameter	Description
IP	The IP address to be resolved
Domains	Domain names (separated by spaces) <b>Example:</b> <code>example.com test.example.com</code>

## Node Selectors

Specify which nodes CoreDNS pods should run on.

Parameter	Description
Label Key	The label key to match on nodes
Label Value	The label value to match on nodes

## Node Tolerations

Allow CoreDNS pods to be scheduled on nodes with taints.

Parameter	Description
Key	The taint key to tolerate
Value	The taint value (optional)
Type	<code>NoSchedule</code> , <code>PreferNoSchedule</code> , or <code>NoExecute</code>

4. Click **Update** to apply the configuration.

# How To

---

## Tasks for Ingress-Nginx

### Tasks for Ingress-Nginx

Prerequisites

Max Connections

Request Timeout

Session Affinity (Sticky Sessions)

Header Modification

URL Rewrite

HSTS (HTTP Strict Transport Security)

Rate Limiting

WAF

Forward-header control

HTTPS

Preserve Source IP

---

## Tasks for Envoy Gateway

### Tasks for Envoy Gateway

[Prerequisites](#)

[Introduction](#)

[Common Tasks For Route Config](#)

[Advanced Configuration](#)

[More configuration](#)

---

## Soft Data Center LB Solution (Alpha)

### Soft Data Center LB Solution (Alpha)

[Prerequisites](#)

[Procedure](#)

[Verification](#)

---

## Kube OVN

### Understanding Kube-OVN CNI

[Upstream OVN/OVS Components](#)

[Core Controller and Agent](#)

[Monitoring, Operation and Maintenance Tools](#)

### Preparing Kube-OVN Underlay

[Usage Instructions](#)

[Terminology Explanation](#)

[Environment Requirements](#)

[Configuration Example](#)

### Automatic I

[Procedure](#)

### Kube-OVN I

[Overview](#)

### Configure Egress Gateway

About Egress Gateway

Notes

Usage

Configuration Parameters

Additional resources

clus

## Configuring (Alpha)

Installing Multus

Creating Subne

id w

ial M

feat

Relevant operations

## Configure IPPool

Instructions

Precautions

## Configure MTU

Default MTU Behavior

Customizing MTU Settings

---

# Configure Endpoint Health Checker

## Configure Endpoint Health Checker

Overview

Key Features

Installation

How It Works

How To Activate

Uninstallation

# alb

## Tasks for ALB

[How To Set NodeSelector And Tolerations For alb-operator](#)

[How To Set NodeSelector And Tolerations For alb](#)

# Tasks for Ingress-Nginx

---

## TOC

### Prerequisites

Max Connections

Request Timeout

Session Affinity (Sticky Sessions)

Header Modification

URL Rewrite

HSTS (HTTP Strict Transport Security)

Rate Limiting

WAF

Forward-header control

HTTPS

- TLS re-encrypt and verify backend certificate

- TLS edge termination

- Passthrough

- Default Certificate

- Add Pod Annotation in IngressNginx

Preserve Source IP

- Via HAProxy Proxy Protocol

  - How it works

  - How to configure

- Via MetalLB with externalTrafficPolicy=Local

  - How it works

How to configure

---

## Prerequisites

[Install ingress-nginx](#)

## Max Connections

[Max-Worker-Connections](#) ↗

## Request Timeout

[Configure request timeout](#) ↗

## Session Affinity (Sticky Sessions)

[Configure sticky sessions](#) ↗

## Header Modification

action	link
set header in request	<a href="#">proxy-set-header</a> ↗
remove header in request	set a empty header in request
set header in response	<a href="#">configuration-snippets</a> ↗ with <a href="#">more-set-header</a> ↗ directive
remove header in response	<a href="#">hide-headers</a> ↗

## URL Rewrite

[rewrite ↗](#)

## HSTS (HTTP Strict Transport Security)

[configure HSTS ↗](#)

## Rate Limiting

[config rate limiting ↗](#)

## WAF

[modsecurity ↗](#)

## Forward-header control

[x-forwarded-prefix-header ↗](#)

## HTTPS

### TLS re-encrypt and verify backend certificate

[verify backend https certificate ↗](#)

### TLS edge termination

[backend protocol ↗](#)

## Passthrough

[ssl-passthrough](#) ↗

## Default Certificate

[default-ssl-certificate](#) ↗

## Add Pod Annotation in IngressNginx

[Add pod annotation](#)

## Preserve Source IP

When traffic passes through load balancers or proxies, the original client IP address can be lost due to NAT (Network Address Translation). Preserving the source IP is important for:

- Access control and security policies
- Accurate logging and analytics
- Rate limiting per client
- Geolocation-based routing

## Via HAProxy Proxy Protocol

### How it works

The [PROXY protocol](#) ↗ is a network protocol for preserving client connection information when proxying TCP connections. It works by prepending a header to the TCP connection that contains the original source IP and port.

#### Traffic flow:

1. Client connects to HAProxy load balancer
2. HAProxy prepends PROXY protocol header with original client IP to the connection

3. Ingress-nginx receives the connection and parses the PROXY protocol header
4. Ingress-nginx extracts the real client IP from the header
5. Backend applications receive the correct client IP in `X-Forwarded-For` and `X-Real-IP` headers

### Advantages:

- Works with any load balancer that supports PROXY protocol (HAProxy, AWS NLB, etc.)
- Preserves source IP across multiple proxy layers
- No impact on routing or node selection

### Considerations:

- Both the load balancer and Ingress-nginx must be configured to use PROXY protocol
- All traffic to Ingress-nginx must use PROXY protocol once enabled (mixing PROXY and non-PROXY traffic will cause connection failures)

## How to configure

Configure your HAProxy load balancer to send PROXY protocol headers, then deploy an ingress-nginx with proxy-protocol support enabled:

```
apiVersion: ingress-nginx.alauda.io/v1
kind: IngressNginx
metadata:
  name: demo
  namespace: ingress-nginx-operator
spec:
  controller:
    config:
      use-proxy-protocol: "true" # enable proxy-protocol support
```

```
frontend tcp_front_80
  bind *:80
  mode tcp
  default_backend ingress_tcp_80

frontend tcp_front_443
  bind *:443
  mode tcp
  default_backend ingress_tcp_443

backend ingress_tcp_80
  mode tcp
  balance roundrobin
  server node1 192.168.133.46:80 check send-proxy-v2

backend ingress_tcp_443
  mode tcp
  balance roundrobin
  server node1 192.168.133.46:443 check send-proxy-v2
```

For more details, see [PROXY protocol documentation](#) ↗.

**Note:** HAProxy can use TCP mode to forward traffic without handling TLS certificates. Since the PROXY protocol works at the TCP layer, you can let Ingress-Nginx handle HTTPS termination and certificate management directly, eliminating the need to configure certificates in HAProxy.

## Via MetalLB with externalTrafficPolicy=Local

### How it works

When using a Kubernetes Service with `type: LoadBalancer`, the default behavior (`externalTrafficPolicy: Cluster`) performs source NAT, which replaces the client IP with the node's IP. Setting `externalTrafficPolicy: Local` preserves the source IP by:

1. **Direct routing:** Traffic is only routed to pods on the same node that received the traffic
2. **No SNAT:** The kube-proxy does not perform source NAT, preserving the original client IP
3. **Health checks:** Only nodes with healthy local pods are included in the load balancer pool

## Traffic flow:

1. Client connects to MetalLB virtual IP
2. MetalLB routes traffic directly to a node with Ingress-NGinx pods
3. Traffic goes directly to the local Ingress-NGinx pod without SNAT
4. Ingress-NGinx sees the real client IP
5. Backend applications receive the correct client IP in headers

## Advantages:

- Simple configuration, no additional protocol required
- Native Kubernetes feature
- Lower latency (no extra proxy hop)

## Considerations:

- **Uneven load distribution:** Traffic can only go to nodes with local pods, potentially causing imbalanced load
- **Pod scheduling:** Ingress-NGinx pods must be scheduled on nodes that MetalLB can route to (use nodeSelector to ensure alignment)
- **Health check behavior:** If all local pods are unhealthy, the node is removed from load balancing entirely

## How to configure

Deploy an ingress-nginx with `externalTrafficPolicy: Local` and ensure pod placement aligns with MetalLB configuration:

```
apiVersion: ingress-nginx.alauda.io/v1
kind: IngressNginx
metadata:
  name: demo
  namespace: ingress-nginx-operator
spec:
  controller:
    service:
      type: LoadBalancer # Use MetalLB to provision a
LoadBalancer service
      externalTrafficPolicy: Local # Preserve source IP by routi
ng traffic only to local pods
      annotations:
        metallb.universe.tf/address-pool: demo-pool # Specify the Metall
B IP address pool to use
      nodeSelector: # Schedule pods only on nodes
matching these labels. This selector must match the MetalLB address poo
l's node selector
        ingress-nginx: "true"
```

**Important:** The `nodeSelector` must match the nodes in your MetalLB address pool configuration to ensure Ingress-Nginx pods are scheduled on nodes that can receive traffic from MetalLB.

For more details, see [externalTrafficPolicy documentation](#).

---

# Tasks for Envoy Gateway

---

## TOC

### Prerequisites

Introduction

HTTP/TCP/UDP Route

PolicyAttachment Via TargetRefs

Global Configuration

Common Tasks For Route Config

Advanced Configuration

OpenTelemetry(Otel)

How To Attach to Listener Created In Other Namespace

How To Use Cert Created In Other Namespace

How To Use SSL passthrough

How To Change SSL Cipher

How To Specify NodePort When Using NodePort Service

How To Add Pod Annotation in EnvoyGateway

How To Set NodeSelector And Tolerations For envoy-gateway-operator

How To Set NodeSelector And Tolerations For envoy-gateway

How To Set NodeSelector And Tolerations For envoy-proxy

More configuration

---

# Prerequisites

1. [Configure EnvoyGatewayCtl](#)
2. [Configure Gateway](#)
3. [Configure Route](#)

## Introduction

When applying configuration changes in the Gateway API, there are three primary approaches available:

1. Direct modification of the HTTP/TCP/UDP Route or `Gateway` .
2. Modification through PolicyAttachment provided by `Gateway Api` and `Envoy Gateway` .
3. Modification on the global configuration level of the `envoy-gateway instance` .

## HTTP/TCP/UDP Route

- [HTTPRoute specification](#) ↗
- [TCPRoute specification](#) ↗
- [UDPRoute specification](#) ↗

## PolicyAttachment Via TargetRefs

Envoy Gateway provides a rich custom policy mechanism that can be attached to gateway resources through the Gateway API's [PolicyAttachment model](#) ↗ .

Envoy Gateway policies are divided into multiple types, including security policies, traffic management policies and more. These policies can be applied to different levels of resources, such as Gateway, HTTPRoute, or Service.

The Gateway API's PolicyAttachment mechanism allows users to attach policies to gateway resources in a declarative way. This mechanism is implemented through the `targetRefs`

field, which specifies the target resource for policy application. For example, policies can be attached to specific Gateways, HTTPRoutes, or Services.

Policy types supported by Envoy Gateway include:

Policy Type	Description
<a href="#">ClientTrafficPolicy</a> ↗	Configuration related to the client-to-proxy communication path, including parameters such as timeouts, retries, keepalive settings, etc.
<a href="#">BackendTrafficPolicy</a> ↗	Configuration related to the proxy-to-backend communication path, including parameters such as timeouts, retries, keepalive settings, etc.
<a href="#">SecurityPolicy</a> ↗	Configuration related to security mechanisms and controls, such as authentication and authorization.

Using the PolicyAttachment mechanism, users can flexibly add, modify, or delete policies without modifying core resource definitions, achieving separation of concerns and better resource management.

## Global Configuration

The configuration related to `envoy-gateway instance` itself or global-level configuration related to all gateways belongs to this `envoy-gateway instance`, such as deployment mode or backend routing.

We recommend using [EnvoyGatewayCtl](#) to manage those global configurations.

## Common Tasks For Route Config

Feature	CR	Description
Auth	envoygateway:SecurityPolicy	<a href="#">Authorization</a> ↗
CORS	gatewayapi:HTTPRoute	<a href="#">Cross-Origin Resource Sharing</a> ↗

Feature	CR	Description
Header Modification	gatewayapi:HTTPRoute	<a href="#">HTTP Header Modification ↗</a>
HTTP Redirect	gatewayapi:HTTPRoute	<a href="#">HTTP Redirect ↗</a>
L7 Timeout	gatewayapi:HTTPRoute	<a href="#">Request Timeouts ↗</a>
SessionAffinity	gatewayapi:HTTPRoute	<a href="#">Session Affinity/Sticky Sessions ↗</a>
L7 Keepalive	envoygateway:ClientTrafficPolicy	<a href="#">L7 Keepalive Timeout Settings ↗</a>
L4 Keepalive	envoygateway:ClientTrafficPolicy	<a href="#">L4 TCP Keepalive Settings ↗</a>
UrlRewrite	gatewayapi:HTTPRoute	<a href="#">URL Rewrite ↗</a>
Retry	gatewayapi:HTTPRoute or envoygateway:BackendTrafficPolicy	<a href="#">Request Retries Config Via HTTPRoute ↗</a> <a href="#">Request Retries Config Via EnvoyGateway ↗</a>
GZip	envoygateway:BackendTrafficPolicy	<a href="#">HTTP Compression ↗</a>

## Advanced Configuration

### OpenTelemetry(Otel)

Please follow instructions in [OpenTelemetry Integration ↗](#) , but use `EnvoyGatewayCtl` to modify the `envoy-gateway-config` .

### How To Attach to Listener Created In Other Namespace

In the Gateway's listener configuration, you need to specify which namespaces are allowed to attach Routes to it.

```
apiVersion: gateway.networking.k8s.io/v1
kind: Gateway
metadata:
  name: example-gateway
spec:
  listeners:
    - name: http-80
      protocol: HTTP
      port: 80
      allowedRoutes:
        namespaces:
          from: All # without limit
    - name: http-81
      protocol: HTTP
      port: 81
      allowedRoutes:
        namespaces:
          from: Same # only allow routes in the same namespace
    - name: http-82
      protocol: HTTP
      port: 82
      allowedRoutes:
        namespaces:
          from: Selector
          selector:
            matchLabels:
              team: frontend # only allow routes in the namespace with la
bel team=frontend
```

Please refer to [Cross-Namespace routing](#) for more details.

## How To Use Cert Created In Other Namespace

To use a certificate created in another namespace, you need to create a `ReferenceGrant` in the namespace where the certificate is created. Please follow instructions in [cross-namespace-certificate-references](#) and [referencegrant](#).

**NOTE**

You cannot specify individual `secret` resources; you must allow the entire namespace

## How To Use SSL passthrough

Please follow instructions in

- [tls](#)
- [tls-passthrough](#)

## How To Change SSL Cipher

Please follow instructions in [customize-gateway-tls-parameters](#)

```
cat <<EOF | kubectl apply -f -
apiVersion: gateway.envoyproxy.io/v1alpha1
kind: ClientTrafficPolicy
metadata:
  name: enforce-tls-13
  namespace: default
spec:
  targetRefs:
  - group: gateway.networking.k8s.io
    kind: Gateway
    name: eg
  tls:
    minVersion: "1.3"
EOF
```

the `.spec.tls` in `ClientTrafficPolicy` is [clienttlssettings](#)

## How To Specify NodePort When Using NodePort Service

When using a NodePort service, kubernetes assigns a NodePort port number to each service port. When accessing the service through a node IP, you should use the corresponding

NodePort port number rather than the service port.

There are two approaches to handle this:

Manually retrieve the NodePort assignment by following [get nodeport from svc port](#)

Manually specify the NodePort in the `EnvoyProxy` configuration instead of letting Kubernetes automatically assign it.

```

apiVersion: gateway.envoyproxy.io/v1alpha1
kind: EnvoyProxy
metadata:
  name: demo
spec:
  ipFamily: DualStack
  provider:
    kubernetes:
      envoyDeployment:
        container:
          imageRepository: registry.alauda.cn:60080/acp/envoyproxy/envoy
      envoyService:
        patch: 1
        type: StrategicMerge
        value:
          spec:
            ports:
              - nodeport: 31888
                port: 80
            type: NodePort
        type: Kubernetes

```

- 1 Use patch field to patch the generated service resource to specify the NodePort

## NOTE

NodePort can only be within a specific range, typically `30000-32767`. If you want the Gateway listener port and NodePort to be consistent, your listener port must also be within the NodePort range.

# How To Add Pod Annotation in EnvoyGateway

[Add pod annotation](#)

## How To Set NodeSelector And Tolerations For envoy-gateway-operator

update the `Subscription` resources

```
# example of nodeSelector and tolerations
kubectl patch subscription envoy-gateway-operator -n envoy-gateway-opera
tor --type='merge' -p '
{
  "spec": {
    "config": {
      "nodeSelector": {
        "node-role.kubernetes.io/infra": ""
      },
      "tolerations": [
        {
          "effect": "NoSchedule",
          "key": "node-role.kubernetes.io/infra",
          "operator": "Equal",
          "value": "reserved"
        }
      ]
    }
  }
}'
```

## How To Set NodeSelector And Tolerations For envoy-gateway

update the `EnvoyGatewayCtl` resources

```
# in default $NAME=cpaas-default and $NS=envoy-gateway-operator
kubectl patch envoygatewayctl $NAME -n $NS --type='merge' -p '
{
  "spec": {
    "deployment": {
      "pod": {
        "nodeSelector": {
          "node-role.kubernetes.io/infra": ""
        },
        "tolerations": [
          {
            "effect": "NoSchedule",
            "key": "node-role.kubernetes.io/infra",
            "operator": "Equal",
            "value": "reserved"
          }
        ]
      }
    }
  }
}'
```

## How To Set NodeSelector And Tolerations For envoy-proxy

update the `EnvoyProxy` resources

```
kubectl patch envoyproxy $NAME -n $NS --type='merge' -p '{
  "spec": {
    "provider": {
      "kubernetes": {
        "envoyDeployment": {
          "pod": {
            "nodeSelector": {
              "node-role.kubernetes.io/infra": ""
            },
            "tolerations": [
              {
                "effect": "NoSchedule",
                "key": "node-role.kubernetes.io/infra",
                "operator": "Equal",
                "value": "reserved"
              }
            ]
          }
        }
      }
    }
  }
}'
```

## More configuration

Please refer to [EnvoyGateway Tasks](#) ↗

# Soft Data Center LB Solution (Alpha)

Deploy a pure software data center load balancer (LB) by creating a highly available load balancer outside the cluster, providing load balancing capabilities for multiple ALBs to ensure stable business operations. It supports configuration for IPv4 only, IPv6 only, or both IPv4 and IPv6 dual stack.

## TOC

[Prerequisites](#)

[Procedure](#)

[Verification](#)

## Prerequisites

1. Prepare two or more host nodes as LB. It is recommended to install Ubuntu 22.04 operating system on LB nodes to reduce the time for LB to forward traffic to abnormal backend nodes.
2. Pre-install the following software on all host nodes of the external LB (this chapter takes two external LB host nodes as an example):
  - `ipvsadm`
  - `container-runtime` such as `containerd`
3. Ensure that the `container-runtime` starts on boot for each host.
4. Ensure that the clock of each host node is synchronized.

5. Prepare the image for Keepalived, used to start the external LB service; the platform already contains this image. The image address is in the following format: `<image repository address>/tkestack/keepalived:<version suffix>`. The version suffix may vary slightly among different versions. You can obtain the image repository address and version suffix as follows. This document uses `build-harbor.alauda.cn/tkestack/keepalived:v3.16.0-beta.3.g598ce923` as an example.

- In the global cluster, execute `kubectl get prdb base -o json | jq .spec.registry.address` to get the **image repository address** parameter.
- In the directory where the installation package is extracted, execute `cat ./installer/res/artifacts.json |grep keepalived -C 2|grep tag|awk '{print $2}'|awk -F '"' '{print $2}'` to get the **version suffix**.

## Procedure

**Note:** The following operations must be executed once on each external LB host node, and the `hostname` of the host nodes must not be duplicated.

1. Add the following configuration information to the file `/etc/modules-load.d/alive.kmod.conf`.

```
ip_vs
ip_vs_rr
ip_vs_wrr
ip_vs_sh
nf_conntrack_ipv4
nf_conntrack
ip6t_MASQUERADE
nf_nat_masquerade_ipv6
ip6table_nat
nf_conntrack_ipv6
nf_defrag_ipv6
nf_nat_ipv6
ip6_tables
```

2. Add the following configuration information to the file `/etc/sysctl.d/alive.sysctl.conf`.

```
net.ipv4.ip_forward = 1
net.ipv4.conf.all.arp_accept = 1
net.ipv4.vs.contrack = 1
net.ipv4.vs.conn_reuse_mode = 0
net.ipv4.vs.expire_nodest_conn = 1
net.ipv4.vs.expire_quiescent_template = 1
net.ipv6.conf.all.forwarding=1
```

- Restart using the `reboot` command.
- Create a folder for the Keepalived configuration file.

```
mkdir -p /etc/keepalived
mkdir -p /etc/keepalived/kubecfg
```

- Modify the configuration items according to the comments in the following file and save them in the `/etc/keepalived/` folder, naming the file `alive.yaml`.



instances:

```

- vip: # Multiple VIPs can be configured
  vip: 192.168.128.118 # VIPs must be different
  id: 20 # Each VIP's ID must be unique, optional
  interface: "eth0"
  check_interval: 1 # optional, default 1: interval to execute check script
  check_timeout: 3 # optional, default 3: check script timeout period
  name: "vip-1" # Identifier for this instance, can only contain alphanumeric characters and hyphens, cannot start with a hyphen
  peer: [ "192.168.128.116", "192.168.128.75" ] # Keepalived node IP, actual generated keepalived.conf will remove all IPs on the interface.

  kube_lock:
    kubecfgs: # The kube-config list used by kube-lock will sequentially attempt these kubecfgs for leader election in Keepalived
      - "/live/cfg/kubecfg/kubecfg01.conf"
      - "/live/cfg/kubecfg/kubecfg02.conf"
      - "/live/cfg/kubecfg/kubecfg03.conf"
    ipvs: # Configuration for option IPVS
      ips: [ "192.168.143.192", "192.168.138.100", "192.168.129.100" ] # IPVS backend, change k8s master node IP to ALB node's node IP
      ports: # Configure health check logic for each port on the VIP
        - port: 80 # The port on the virtual server must match the real server's port
          virtual_server_config: |
            delay_loop 10 # Interval for performing health checks on the real server
            lb_algo rr
            lb_kind NAT
            protocol TCP
          raw_check: |
            TCP_CHECK {
              connect_timeout 10
              connect_port 1936
            }
- vip:
  vip: 2004::192:168:128:118
  id: 102
  interface: "eth0"
  peer: [ "2004::192:168:128:75", "2004::192:168:128:116" ]
  kube_lock:

```

```

kubecfgs: # The kube-config list used by kube-lock will sequentially attempt these kubecfgs for leader election in Keepalived
  - "/live/cfg/kubecfg/kubecfg01.conf"
  - "/live/cfg/kubecfg/kubecfg02.conf"
  - "/live/cfg/kubecfg/kubecfg03.conf"

ipvs:
  ips: [ "2004::192:168:143:192", "2004::192:168:138:100", "2004::192:168:129:100" ]
  ports:
    - port: 80
  virtual_server_config: |
    delay_loop 10
    lb_algo rr
    lb_kind NAT
    protocol TCP
  raw_check: |
    TCP_CHECK {
      connect_timeout 1
      connect_port 1936
    }

```

6. Execute the following command in the business cluster to check the certificate expiration date in the configuration file, ensuring that the certificate is still valid. The LB functionality will become unavailable after the certificate expires, requiring contact with the platform administrator for a certificate update.

```

openssl x509 -in <(cat /etc/kubernetes/admin.conf | grep client-certificate-data | awk '{print $NF}' | base64 -d ) -noout -dates

```

7. Copy the `/etc/kubernetes/admin.conf` file from the three Master nodes in the Kubernetes cluster to the `/etc/keepalived/kubecfg` folder on the external LB nodes, naming them with an index, e.g., `kubecfg01.conf`, and modify the `apiserver` node addresses in these three files to the actual node addresses of the Kubernetes cluster.

**Note:** After the platform certificate is updated, this step needs to be executed again, overwriting the original files.

8. Check the validity of the certificates.

1. Copy `/usr/bin/kubect1` from the Master node of the business cluster to the LB node.
2. Execute `chmod +x /usr/bin/kubect1` to grant execution permissions.
3. Execute the following commands to confirm certificate validity.

```
kubectl --kubeconfig=/etc/keepalived/kubecfg/kubecfg01.conf get node
kubectl --kubeconfig=/etc/keepalived/kubecfg/kubecfg02.conf get node
kubectl --kubeconfig=/etc/keepalived/kubecfg/kubecfg03.conf get node
```

If the following results are returned, the certificate is valid.

```
kubectl --kubeconfig=/etc/keepalived/kubecfg/kubecfg01.conf get node
## Output
```

NAME	STATUS	ROLES	AGE	VERSION
192.168.129.100	Ready	<none>	7d22h	v1.25.6
192.168.134.167	Ready	control-plane,master	7d22h	v1.25.6
192.168.138.100	Ready	<none>	7d22h	v1.25.6
192.168.143.116	Ready	control-plane,master	7d22h	v1.25.6
192.168.143.192	Ready	<none>	7d22h	v1.25.6
192.168.143.79	Ready	control-plane,master	7d22h	v1.25.6

```
kubectl --kubeconfig=/etc/keepalived/kubecfg/kubecfg02.conf get node
## Output
```

NAME	STATUS	ROLES	AGE	VERSION
192.168.129.100	Ready	<none>	7d22h	v1.25.6
192.168.134.167	Ready	control-plane,master	7d22h	v1.25.6
192.168.138.100	Ready	<none>	7d22h	v1.25.6
192.168.143.116	Ready	control-plane,master	7d22h	v1.25.6
192.168.143.192	Ready	<none>	7d22h	v1.25.6
192.168.143.79	Ready	control-plane,master	7d22h	v1.25.6

```
kubectl --kubeconfig=/etc/keepalived/kubecfg/kubecfg03.conf get node
## Output
```

NAME	STATUS	ROLES	AGE	VERSION
192.168.129.100	Ready	<none>	7d22h	v1.25.6
192.168.134.167	Ready	control-plane,master	7d22h	v1.25.6
192.168.138.100	Ready	<none>	7d22h	v1.25.6
192.168.143.116	Ready	control-plane,master	7d22h	v1.25.6
192.168.143.192	Ready	<none>	7d22h	v1.25.6
192.168.143.79	Ready	control-plane,master	7d22h	v1.25.6

9. Upload the Keepalived image to the external LB node and run Keepalived using nerdctl.

```
nerdctl run -dt --restart=always --privileged --network=host -v /etc/keepalived:/live/cfg build-harbor.alauda.cn/tkestack/keepalived:v3.16.0-beta.3.g598ce923
```

10. Run the following command on the node accessing `keepalived`: `sysctl -w net.ipv4.conf.all.arp_accept=1`.

## Verification

1. Run the command `ipvsadm -ln` to view the IPVS rules, and you will see IPv4 and IPv6 rules applicable to the business cluster ALBs.

```
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port          Forward Weight      ActiveConn InAct
tConn
TCP  192.168.128.118:80 rr
  -> 192.168.129.100:80          Masq    1        0          0
  -> 192.168.138.100:80          Masq    1        0          0
  -> 192.168.143.192:80          Masq    1        0          0
TCP  [2004::192:168:128:118]:80 rr
  -> [2004::192:168:129:100]:80  Masq    1        0          0
  -> [2004::192:168:138:100]:80  Masq    1        0          0
  -> [2004::192:168:143:192]:80  Masq    1        0          0
```

2. Shut down the LB node where the VIP is located and test whether the VIP of both IPv4 and IPv6 can successfully migrate to another node, typically within 20 seconds.
3. Use the `curl` command on a non-LB node to test if communication with the VIP is normal.

```
curl 192.168.128.118
```

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed
and working. Further configuration is required.</p>

<p>For online documentation and support please refer to <a href="htt
p://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at <a href="http://nginx.com/">nginx.co
m</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

```
curl -6 [2004::192:168:128:118]:80 -g
```

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed
and working. Further configuration is required.</p>

<p>For online documentation and support please refer to <a href="htt
p://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at<a href="http://nginx.com/">nginx.com
</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

# Kube OVN

## Understanding Kube-OVN CNI

- Upstream OVN/OVS Components
- Core Controller and Agent
- Monitoring, Operation and Maintenance Tools

## Preparing Kube-OVN Underlay

- Usage Instructions
- Terminology Explanation
- Environment Requirements
- Configuration Example

## Automatic I

- Procedure

## Configure Egress Gateway

- About Egress Gateway
- Notes
- Usage
- Configuration Parameters
- Additional resources

relevant operations

## Configure IPPool

- Instructions
- Precautions

## Configure MTU

## Kube-OVN I

Overview

clus

## Configuring (Alpha)

Installing Multus

Creating Subne

id w

ial M

feat

Default MTU Behavior

Customizing MTU Settings

# Understanding Kube-OVN CNI

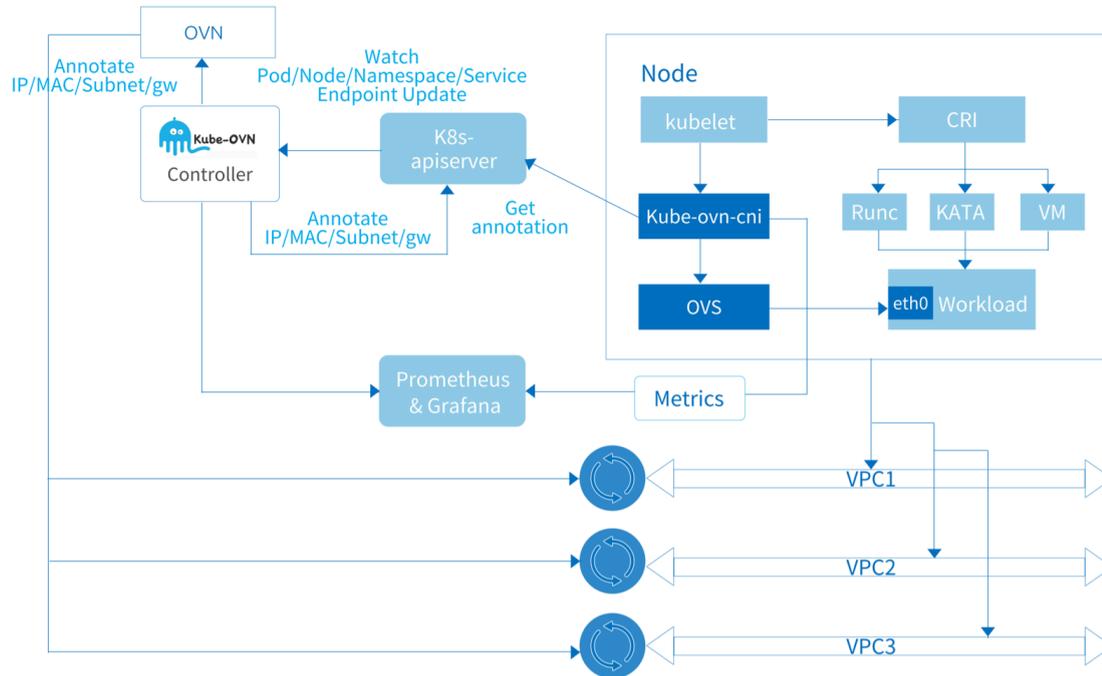
This document describes the general architecture of Kube-OVN, the functionality of each component and how they interact with each other.

Overall, Kube-OVN serves as a bridge between Kubernetes and OVN, combining proven SDN with Cloud Native. This means that Kube-OVN not only implements network specifications under Kubernetes, such as CNI, Service and Networkpolicy, but also brings a large number of SDN domain capabilities to cloud-native, such as logical switches, logical routers, VPCs, gateways, QoS, ACLs and traffic mirroring.

Kube-OVN also maintains a good openness to integrate with many technology solutions, such as Cilium, Submariner, Prometheus, KubeVirt, etc.

The components of Kube-OVN can be broadly divided into three categories.

- Upstream OVN/OVS components.
- Core Controller and Agent.
- Monitoring, operation and maintenance tools and extension components.



## TOC

### Upstream OVN/OVS Components

ovn-central

ovs-ovn

### Core Controller and Agent

kube-ovn-controller

kube-ovn-cni

### Monitoring, Operation and Maintenance Tools and Extension Components

kube-ovn-speaker

kube-ovn-pinger

kube-ovn-monitor

kubectl-ko

## Upstream OVN/OVS Components

This type of component comes from the OVN/OVS community with specific modifications for Kube-OVN usage scenarios. OVN/OVS itself is a mature SDN system for managing virtual machines and containers, and we strongly recommend that users interested in the Kube-OVN implementation read [ovn-architecture\(7\)](#) first to understand what OVN is and how to integrate with it. Kube-OVN uses the northbound interface of OVN to create and coordinate virtual networks and map the network concepts into Kubernetes.

All OVN/OVS-related components have been packaged into images and are ready to run in Kubernetes.

## ovn-central

The `ovn-central` Deployment runs the control plane components of OVN, including `ovn-nb`, `ovn-sb`, and `ovn-northd`.

- `ovn-nb`: Saves the virtual network configuration and provides an API for virtual network management. `kube-ovn-controller` will mainly interact with `ovn-nb` to configure the virtual network.
- `ovn-sb`: Holds the logical flow table generated from the logical network of `ovn-nb`, as well as the actual physical network state of each node.
- `ovn-northd`: translates the virtual network of `ovn-nb` into a logical flow table in `ovn-sb`.

Multiple instances of `ovn-central` will synchronize data via the Raft protocol to ensure high availability.

## ovs-ovn

`ovs-ovn` runs as a DaemonSet on each node, with `openvswitch`, `ovsdb`, and `ovn-controller` running inside the Pod. These components act as agents for `ovn-central` to translate logical flow tables into real network configurations.

## Core Controller and Agent

This part is the core component of Kube-OVN, serving as a bridge between OVN and Kubernetes, bridging the two systems and translating network concepts between them. Most of the core functions are implemented in these components.

## kube-ovn-controller

This component performs the translation of all resources within Kubernetes to OVN resources and acts as the control plane for the entire Kube-OVN system. The `kube-ovn-controller` listens for events on all resources related to network functionality and updates the logical network within the OVN based on resource changes. The main resources listened including:

Pod, [Service](#), Endpoint, Node, [NetworkPolicy](#), VPC, [Subnet](#), [Vlan](#), [ProviderNetwork](#).

Taking the Pod event as an example, `kube-ovn-controller` listens to the Pod creation event, allocates the address via the built-in in-memory IPAM function, and calls `ovn-central` to create logical ports, static routes and possible ACL rules. Next, `kube-ovn-controller` writes the assigned address and subnet information such as CIDR, gateway, route, etc. to the annotation of the Pod. This annotation is then read by `kube-ovn-cni` and used to configure the local network.

## kube-ovn-cni

This component runs on each node as a DaemonSet, implements the CNI interface, and operates the local OVS to configure the local network.

This DaemonSet copies the `kube-ovn` binary to each machine as a tool for interaction between `kubelet` and `kube-ovn-cni`. This binary sends the corresponding CNI request to `kube-ovn-cni` for further operation. The binary will be copied to the `/opt/cni/bin` directory by default.

`kube-ovn-cni` will configure the specific network to perform the appropriate traffic operations, and the main tasks including:

1. Config `ovn-controller` and `vswitchd`.
2. Handle CNI Add/Del requests:
  1. Create or delete veth pair and bind or unbind to OVS ports.

2. Configure OVS ports
3. Update host iptables/ipset/route rules.
3. Dynamically update the network QoS.
4. Create and configure the `ovn0` NIC to connect the container network and the host network.
5. Configure the host NIC to implement Vlan/Underlay/EIP.
6. Dynamically config inter-cluster gateways.

## Monitoring, Operation and Maintenance Tools and Extension Components

These components provide monitoring, diagnostics, operations tools, and external interface to extend the core network capabilities of Kube-OVN and simplify daily operations and maintenance.

### kube-ovn-speaker

This component is a DaemonSet running on a specific labeled nodes that publish routes to the external, allowing external access to the container directly through the Pod IP.

### kube-ovn-pinger

This component is a DaemonSet running on each node to collect OVS status information, node network quality, network latency, etc.

### kube-ovn-monitor

This component collects OVN status information and the monitoring metrics.

### kubectl-ko

This component is a kubectl plugin, which can quickly run common operations.



# Preparing Kube-OVN Underlay Physical Network

The container network under Kube-OVN Underlay transport mode relies on physical network support. Before deploying the Kube-OVN Underlay network, please collaborate with the network administrator to plan and complete the relevant configurations of the physical network in advance, ensuring network connectivity.

## TOC

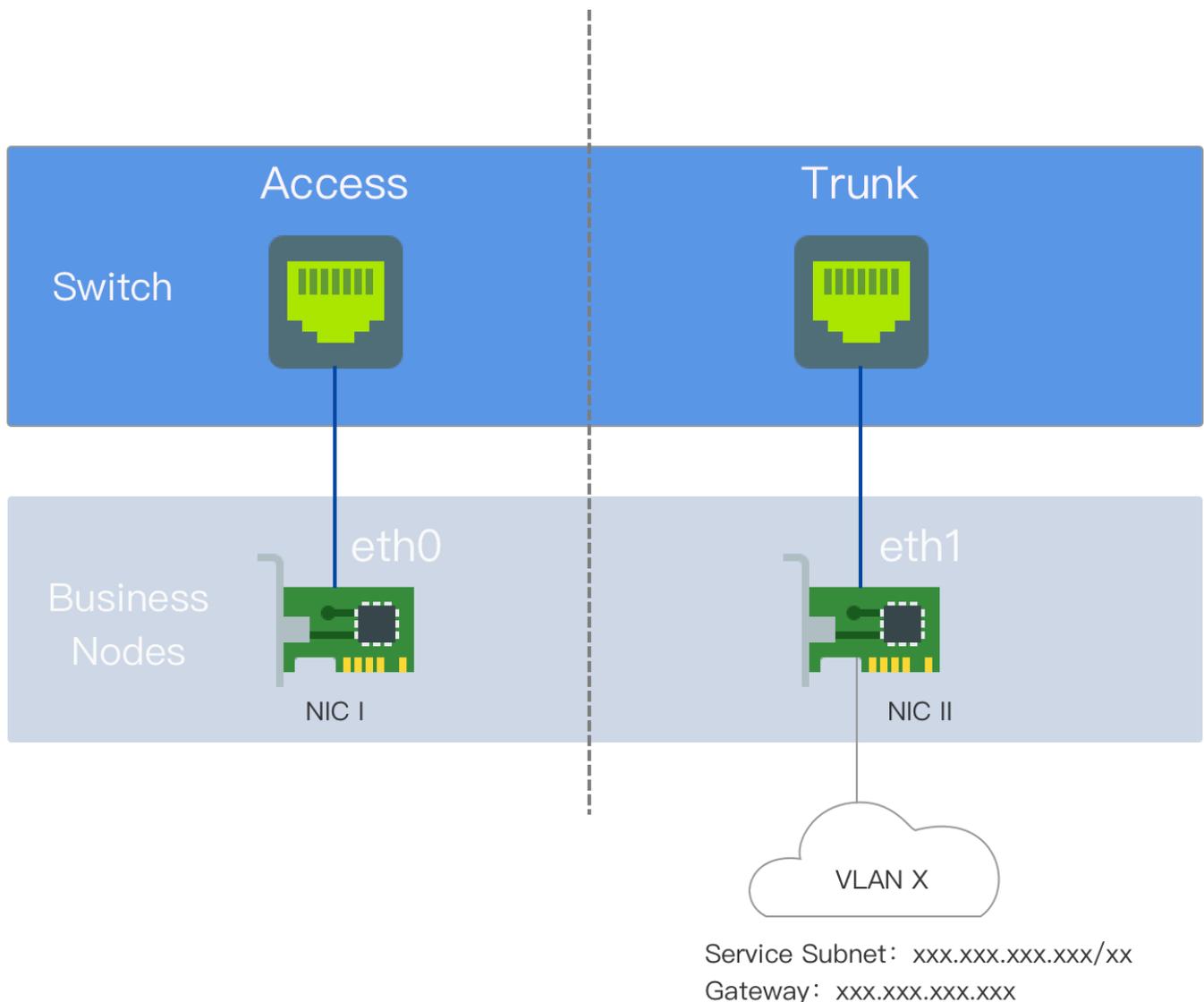
[Usage Instructions](#)[Terminology Explanation](#)[Environment Requirements](#)[Configuration Example](#)[Switch Configuration](#)[Check Network Connectivity](#)[Platform Configuration](#)

## Usage Instructions

Kube-OVN Underlay requires deployment with multiple network interface cards (NICs), and the Underlay subnet must exclusively use one NIC. No other types of traffic, such as SSH, should be on that NIC; they should utilize other NICs.

Before use, ensure that the node server has at least a **dual-NIC** environment, and it is recommended that the NIC speed is **at least 10 Gbps or higher** (e.g., 10 Gbps, 25 Gbps, 40 Gbps).

- NIC One: The NIC with the default route, configured with an IP address, interconnected with the external switch interface, which is set to Access mode.
- NIC Two: The NIC without the default route and not configured with an IP address, interconnected with the external switch interface, which is set to Trunk mode. The Underlay subnet exclusively uses NIC Two.



## Terminology Explanation

VLAN (Virtual Local Area Network) is a technology that logically divides a local area network into multiple segments (or smaller LANs) to facilitate data exchange for virtual workgroups.

The emergence of VLAN technology allows administrators to logically segment different users within the same physical local area network into distinct broadcast domains based on actual application needs. Each VLAN comprises a group of computer workstations with similar requirements and possesses the same properties as a physically formed LAN. Since VLANs are logically divided rather than physically, workstations within the same VLAN are not confined to the same physical area; they can exist across different physical LAN segments.

The main advantages of VLANs include:

- **Port Segmentation.** Even on the same switch, ports in different VLANs cannot communicate with each other. A physical switch can function as multiple logical switches. This is commonly used to control mutual access between different departments and sites in a network.
- **Network Security.** Different VLANs cannot communicate directly, eliminating the insecurity of broadcast information. Broadcast and unicast traffic within a VLAN will not be forwarded to other VLANs, helping control traffic, reduce equipment investments, simplify network management, and improve network security.
- **Flexible Management.** When changing a user's network affiliation, there's no need to replace ports or cables; it merely requires a software configuration change.

## Environment Requirements

In Underlay mode, Kube-OVN bridges a physical NIC to OVS and sends packets directly to the external through that physical NIC. The L2/L3 forwarding capability relies on the underlying network devices. The corresponding gateway, VLAN, and security policies need to be pre-configured on the underlying network devices.

- **Network Configuration Requirements**
  - Kube-OVN checks the gateway's connectivity via ICMP protocol when starting containers; the underlying gateway must respond to ICMP requests.
  - For service access traffic, Pods will first send packets to the gateway, which must have the ability to forward packets back to the local subnet.
  - When the switch or bridge has Hairpin functionality enabled, **Hairpin must be disabled.** If using a VMware virtual machine environment, set

**Net.ReversePathFwdCheckPromisc** on the VMware host to **1**, and Hairpin does not need to be disabled.

- The bridging NIC **cannot** be a **Linux Bridge**.
- NIC bonding modes support Mode 0 (balance-rr), Mode 1 (active-backup), Mode 4 (802.3ad), Mode 6 (balance-alb), with a recommendation to use 0 or 1. Other bonding modes have not been tested; please use them with caution.
- **IaaS (Virtualization) Layer Configuration Requirements**
  - For OpenStack VM environments, the **PortSecurity** for the corresponding network port needs to be disabled.
  - For VMware's vSwitch network, **MAC Address Changes**, **Forged Transmits**, and **Promiscuous Mode Operation** must all be set to **Accept**.
  - For public clouds such as AWS, GCE, and Alibaba Cloud, Underlay mode networks cannot be supported due to their lack of user-defined MAC address capabilities.

## Configuration Example

The nodes in this example are dual-NIC physical machines. NIC One is the NIC with the default route; NIC Two is the NIC without the default route and is not configured with an IP address, exclusively used for the Underlay subnet. NIC Two is interconnected with the external switch.

- On the switch side, the interface connected to NIC Two should be configured in Trunk mode, allowing the corresponding VLANs to pass through.
- Configure the gateway address of the cluster subnet on the corresponding vlan-interface interface. If dual-stack is needed, the IPv6 gateway address can also be configured simultaneously.
- If the gateway is behind a firewall, access from node nodes to the cluster-cidr network must be permitted.
- No configuration is needed for server NICs.

## Switch Configuration

Configure the VLAN Interface:

```
#
interface Vlan-interface74
    ip address 192.168.74.254 255.255.255.0 //IPv4 gateway address
    ipv6 address 2074::192:168:74:254/64 //IPv6 gateway address
#
```

Configure the interface connected to NIC Two:

```
#
interface Ten-GigabitEthernet1/0/19
    port link mode bridge
    port link-type trunk // Configure the interface to Trunk mode
    undo port trunk permit vlan 1
    port trunk permit vlan 74 // Allow the corresponding VLAN to pass through
#
```

## Check Network Connectivity

Test if NIC Two can communicate with the gateway address:

```
ip link add ens224.74 link ens224 type vlan id 74 // The NIC name is ens224, and the VLAN ID is 74
ip link set ens224.74 up
ip addr add 192.168.74.200/24 dev ens224.74 // Select a test address within the Underlay subnet, here it's 192.168.74.200/24
ping 192.168.74.254 // If able to ping the gateway, it confirms that the physical environment meets deployment requirements
ip addr del 192.168.74.200/24 dev ens224.74 // Delete the test address after testing
ip link del ens224.74 // Delete the sub-interface after testing
```

## Platform Configuration

In the left navigation bar, click **Cluster Management > Cluster**, then click **Create Cluster**. For specific configuration procedures, please refer to the [Create Cluster](#) document, with container network configuration shown in the image below.

**Note:** The Join subnet has no practical significance in the Underlay environment and primarily serves to create an Overlay subnet later, providing the IP address range necessary for communication between nodes and container groups.

### Container Networking

IPv4 / IPv6 Dual Stack:

Ensure that all nodes are correctly configured with IPv6 network addresses when enabling IPv4/IPv6 dual stack, as the cluster will not revert to IPv4 single stack after creation.

Network Type: Kube-OVN Calico Flannel Custom ?

Default Subnet:

\* IPv4:  .  .  .  /  IPv4 subnet address of NIC II

\* IPv6:  IPv6 subnet address of NIC II

Transmit Mode: Underlay Overlay ?

Gateway: \* IPv4  IPv4 gateway address \* IPv6  IPv6 gateway address

The default gateway IPv4/IPv6 value must be within the cluster CIDR address range

\* VLAN ID:  VLAN ID that the switch allows to pass through

Preserved IP:

Protocol stack	IP Format	* IP Address
<span style="color: orange;">!</span> If the IP in the subnet is occupied by the physical network, the cluster cannot be created successfully. Please set it as reserved IP		
<span style="color: #007bff;">+</span> Add		

After the cluster is created, new subnets are supported.

\* Service CIDR:

\* IPv4:  .  .  .  /  Custom SVC, must not duplicate with the internal network

\* IPv6:

\* Join CIDR:

\* IPv4: Custom  Address segment of the NIC used for communication on the Overlay network

\* IPv6:

# Automatic Interconnection of Underlay and Overlay Subnets

If a cluster has both Underlay and Overlay subnets, by default, Pods under the Overlay subnet can access Pods' IPs in the Underlay subnet through a gateway using NAT. However, Pods in the Underlay subnet need to configure node routing to access Pods in the Overlay subnet.

To achieve automatic interconnection between Underlay and Overlay subnets, you can manually modify the YAML file of the Underlay subnet. Once configured, Kube-OVN will also use an additional Underlay IP to connect the Underlay subnet and the ovn-cluster logical router, setting the corresponding routing rules to enable interconnection.

---

## TOC

[Procedure](#)

---

## Procedure

1. Go to **Administrator**.
2. In the left navigation bar, click on **Cluster Management > Resource Management**.
3. Enter **Subnet** to filter resource objects.
4. Click on **⋮ > Update** next to the Underlay subnet to be modified.
5. Modify the YAML file, adding the field `u2oInterconnection: true` in the `Spec`.

6. Click **Update**.

**Note:** Existing compute components in the Underlay subnet need to be recreated for the changes to take effect.

---

# Kube-OVN Underlay + MetalLB LoadBalancer Service Configuration

---

## TOC

### Overview

#### Prerequisites

- Environment Requirements

- Traffic Flow

#### Configuration Steps

1. Configure ProviderNetwork with VLAN Sub-interfaces
2. Configure Kube-OVN Controller Parameters
3. Configure Underlay Subnet External Address Feature
4. Create MetalLB External Address Pool
5. Create Sample Application and LoadBalancer Service
6. Verify Configuration
7. Migrating Existing Services

---

## Overview

This solution addresses the integration of MetalLB L2 mode with Kube-OVN Underlay networking. It allows users to utilize Underlay subnet IPs as MetalLB LoadBalancer Service VIPs, directly forwarding traffic to backend business Pods.

---

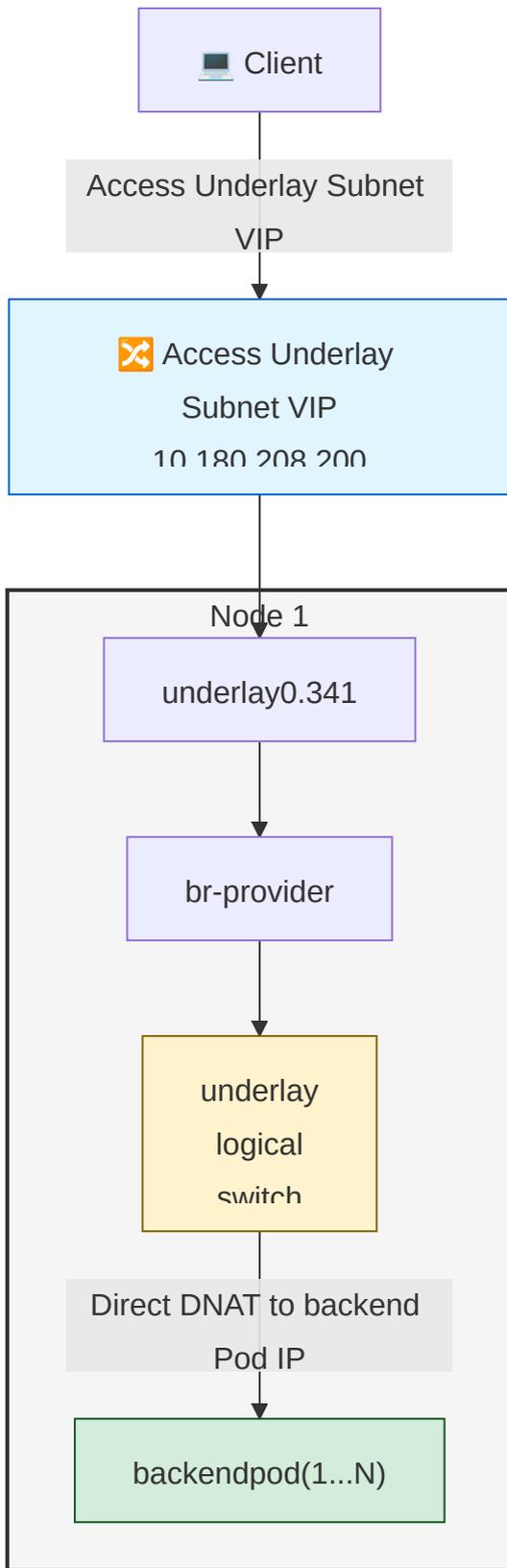
# Prerequisites

## Environment Requirements

- **ACP version:**  $\geq 4.2.2$
- **IP support:** IPv4 only (IPv6 is currently not supported)

## Traffic Flow

**Traffic Diagram:**



## Configuration Steps

# 1. Configure ProviderNetwork with VLAN Sub-interfaces

**Important:** VLAN sub-interfaces must be used.

Configure Kube-OVN Underlay network to automatically create VLAN sub-interfaces:

```
apiVersion: kubeovn.io/v1
kind: ProviderNetwork
metadata:
  name: provider
spec:
  defaultInterface: underlay0.341
  autoCreateVlanSubinterfaces: true # Automatically creates VLAN sub-int
erfaces (e.g., underlay0.341) if only parent interface (underlay0) exists

---
apiVersion: kubeovn.io/v1
kind: Vlan
metadata:
  name: ovn-vlan
spec:
  id: 0 # Use 0 because autoCreateVlanSubinterfaces creates the VLAN s
ub-interface (underlay0.341) which handles VLAN tagging, not Kube-OVN dir
ectly
  provider: provider
status:
  subnets:
  - ovn-default
```

**⚠ Warning:** When modifying the `ProviderNetwork` or `Vlan` resources individually, the Underlay network connectivity will be interrupted. Network connectivity will only be restored after both resources are fully configured and in sync. Plan configuration changes during maintenance windows to minimize service disruption.

## 2. Configure Kube-OVN Controller Parameters

Configure the Kube-OVN controller with the required parameters for LoadBalancer functionality:

## Using Web Console:

1. Navigate to **Administrator > Marketplace > Cluster Plugins**, then search for `ovn` to locate **Alauda Container Platform Networking for Kube-OVN**
2. In the plugin row, click the action menu (vertical `:`) and select **Update** to open the configuration dialog
3. Configure the following settings:
  - **Skip CT for Dst LPort IPs: No**
  - **Enable OVN LB Local: Yes**

## 3. Configure Underlay Subnet External Address Feature

Edit the Underlay subnet to reserve an IP range for LoadBalancer usage:

**Important:** External address pool IPs must be within the Underlay subnet.

Modify the Underlay subnet parameter `spec.enableExternalLBAddress: true`:

```
apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
  name: underlay-subnet
spec:
  enableExternalLBAddress: true      # Indicates this subnet has IP range
  # for LB service VIP
  excludeIps:
  - 10.180.208.200..10.180.208.220  # Reserve IP range for external address pool
```

## 4. Create MetalLB External Address Pool

```
# underlay-ippool.yaml
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: acp-underlay-pool
  namespace: metallb-system
spec:
  addresses:
    - 10.180.208.200-10.180.208.220 # Underlay subnet IP range
  avoidBuggyIPs: true
  autoAssign: true
---
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: acp-underlay-pool
  namespace: metallb-system
spec:
  ipAddressPools:
    - acp-underlay-pool
  nodeSelectors: []
```

Deploy the address pool:

```
kubectl apply -f underlay-ippool.yaml
```

## 5. Create Sample Application and LoadBalancer Service

```
# application-with-loadbalancer.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend-app
  labels:
    app: backend
spec:
  replicas: 3
  selector:
    matchLabels:
      app: backend
  template:
    metadata:
      labels:
        app: backend
    spec:
      containers:
        - name: backend
          image: nginx:1.25
          ports:
            - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: backend-lb-service
  # To use specific IPPool: add annotation `metallb.io/address-pool: acp-
  # underlay-pool`
  # To use fixed IP: set `spec.loadBalancerIP: 10.180.208.201`
spec:
  type: LoadBalancer
  externalTrafficPolicy: Local # **IMPORTANT**: Required for preserving
  source IP and enabling direct Pod routing
  selector:
    app: backend
  ports:
    - port: 80
      targetPort: 80
```

Deploy the application:

```
kubectl apply -f application-with-loadbalancer.yaml
```

## 6. Verify Configuration

```
# Check service status
kubectl get svc backend-lb-service -o wide

# Test external access
curl http://10.180.208.200
```

## 7. Migrating Existing Services

For existing services using the old address pool (node subnet only), you can migrate them to the new Underlay address pool:

```
# Add annotation to migrate existing service
kubectl annotate service <existing-service-name> metallb.io/address-pool=
acp-underlay-pool --overwrite

# Verify the service has been assigned a new IP from the Underlay pool
kubectl get svc <existing-service-name> -o wide
```

For **new services**, add the annotation directly:

```
apiVersion: v1
kind: Service
metadata:
  name: backend-lb-service
  annotations:
    metallb.io/address-pool: acp-underlay-pool # Use the Underlay address pool
spec:
  type: LoadBalancer
  externalTrafficPolicy: Local
  selector:
    app: backend
  ports:
    - port: 80
      targetPort: 80
```

```
# Check service status
kubectl get svc backend-lb-service -o wide

# Test external access
curl http://10.180.208.200
```

# Cluster Interconnection (Alpha)

It supports configuration of cluster interconnection between clusters whose network mode is the same as Kube-OVN, so that Pods in the clusters can access each other. Cluster Interconnect Controller is an extension component provided by Kube-OVN, which is responsible for collecting network information between different clusters and connecting the networks of multiple clusters by issuing routes.

## TOC

### Prerequisites

Multi-node Kube-OVN connectivity controller was built

- Deploy Deployment

- Podman and Containerd Deployment

Deploy the cluster interconnection controller in the Global cluster

Join the cluster interconnect

Relevant operations

- Update the gateway node information of the interconnected cluster

- Exit cluster interconnection

- Cleaning up Interconnected Cluster Residue

- Uninstalling the Interconnected Cluster

- Configure Cluster Gateway High Availability

## Prerequisites

- The subnet CIDRs of different clusters cannot overlap each other.
- There needs to be a set of machines that can be accessed over IP by each cluster's kube-ovn-controller to deploy controllers that interconnect across clusters.
- A set of machines that can be accessed by kube-ovn-controller per cluster via IP for cross-cluster interconnections needs to exist for each cluster to be used as gateway nodes afterward.
- This feature is only available for the default VPC, user-defined VPCs cannot use the interconnect feature.

## Multi-node Kube-OVN connectivity controller was built

There are three deployment methods available: Deploy deployment (supported in platform v3.16.0 and later versions), Podman deployment, and Containerd deployment.

### Deploy Deployment

**Note:** This deployment method is supported in platform v3.16.0 and later versions.

#### Operation Steps

1. Execute the following command on the cluster Master node to obtain the install-ic-server.sh installation script.

```
wget https://github.com/kubeovn/kube-ovn/blob/release-1.12/dist/images/  
install-ic-server.sh
```

2. Open the script file in the current directory and modify the parameters as follows.

```
REGISTRY="kubeovn"  
VERSION=""
```

Modified parameter configurations are as follows:

```
REGISTRY="<Kube-OVN image repository address>" ## For example: REGISTRY="registry.alauda.cn:60080/acp/"  
VERSION="<Kube-OVN version>" ## For example: VERSION="v1.9.25"
```

3. Save the script file and execute it using the following command.

```
sh install-ic-server.sh
```

## Podman and Containerd Deployment

1. Select **three or more nodes in any cluster** to deploy the Interconnected Controller. In this example, three nodes are prepared.
2. Choose any node as the Leader and execute the following commands according to the different deployment methods.

**Note:** Before configuration, please check if there is an ovn directory under `/etc`. If not, use the command `mkdir /etc/ovn` to create one.

- **Commands for container deployment Note:** Execute the command `podman images | grep ovn` to obtain the Kube-OVN image address.
  - Command for the Leader node:

```

podman run \
--name=ovn-ic-db \
-d \
--env "ENABLE_OVN_LEADER_CHECK=false" \
--network=host \
--restart=always \
--privileged=true \
-v /etc/ovn:/etc/ovn \
-v /var/run/ovn:/var/run/ovn \
-v /var/log/ovn:/var/log/ovn \
-e LOCAL_IP="<IP address of the current node>" \   ## For example:
-e LOCAL_IP="192.168.39.37"
-e NODE_IPS="<IP addresses of all nodes, separated by commas>" \
## For example: -e NODE_IPS="192.168.39.22,192.168.39.24,192.168.3
9.37"
<image repository address> bash start-ic-db.sh   ## For example:
192.168.39.10:60080/acp/kube-ovn:v1.8.8 bash start-ic-db.sh

```

- Commands for the other two nodes:

```

podman run \
--name=ovn-ic-db \
-d \
--env "ENABLE_OVN_LEADER_CHECK=false" \
--network=host \
--restart=always \
--privileged=true \
-v /etc/ovn:/etc/ovn \
-v /var/run/ovn:/var/run/ovn \
-v /var/log/ovn:/var/log/ovn \
-e LOCAL_IP="<IP address of the current node>" \   ## For example:
-e LOCAL_IP="192.168.39.24"
-e LEADER_IP="<IP address of the Leader node>" \   ## For example:
-e LEADER_IP="192.168.39.37"
-e NODE_IPS="<IP addresses of all nodes, separated by commas>" \
## For example: -e NODE_IPS="192.168.39.22,192.168.39.24,192.168.3
9.37"
<image repository address> bash start-ic-db.sh   ## For example: 1
92.168.39.10:60080/acp/kube-ovn:v1.8.8 bash start-ic-db.sh

```

- **Commands for Containerd deployment**

**Note:** Execute the command `crictl images | grep ovn` to obtain the Kube-OVN image address.

- Command for the Leader node:

```
ctr -n k8s.io run \  
-d \  
--env "ENABLE_OVN_LEADER_CHECK=false" \  
--net-host \  
--privileged \  
--mount="type=bind,src=/etc/ovn/,dst=/etc/ovn,options=rbind:rw" \  
--mount="type=bind,src=/var/run/ovn,dst=/var/run/ovn,options=rbind:rw" \  
--mount="type=bind,src=/var/log/ovn,dst=/var/log/ovn,options=rbind:rw" \  
--env="NODE_IPS=<IP addresses of all nodes, separated by commas>" \  
\  ## For example: --env="NODE_IPS="192.168.178.97,192.168.181.93,192.168.177.192"" \  
--env="LOCAL_IP=<IP address of the current node>" \  ## For example: --env="LOCAL_IP="192.168.178.97"" \  
<image repository address> ovn-ic-db bash start-ic-db.sh  ## For example: registry.alauda.cn:60080/acp/kube-ovn:v1.9.25 ovn-ic-db bash start-ic-db.sh
```

- Commands for the other two nodes:

```

ctr -n k8s.io run \
-d \
--env "ENABLE_OVN_LEADER_CHECK=false" \
--net-host \
--privileged \
--mount="type=bind,src=/etc/ovn/,dst=/etc/ovn,options=rbind:rw" \
--mount="type=bind,src=/var/run/ovn,dst=/var/run/ovn,options=rbind:rw" \
--mount="type=bind,src=/var/log/ovn,dst=/var/log/ovn,options=rbind:rw" \
--env="NODE_IPS=<IP addresses of all nodes, separated by commas>" \
\  ## For example: --env="NODE_IPS="192.168.178.97,192.168.181.93,192.168.177.192"" \
--env="LOCAL_IP=<IP address of the current node>" \  ## For example: --env="LOCAL_IP="192.168.181.93""
--env="LEADER_IP=<IP address of the Leader node>" \  ## For example: --env="LEADER_IP="192.168.178.97""
<image repository address> ovn-ic-db bash start-ic-db.sh  ## For example: registry.alauda.cn:60080/acp/kube-ovn:v1.9.25 ovn-ic-db bash start-ic-db.sh

```

## Deploy the cluster interconnection controller in the Global cluster

In any control node of global, replace the following parameters according to the comments and execute the following command to create the ConfigMap resource.

**Note:** To ensure the correct operation, the ConfigMap named ovn-ic on global is not allowed to be modified. If any parameter needs to be changed, please delete the ConfigMap and reconfigure it correctly before applying the ConfigMap.

```
cat << EOF | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  name: ovn-ic
  namespace: cpaas-system
data:
  ic-db-host: "192.168.39.22,192.168.39.24,192.168.39.37" # Address of
the node where the cluster interconnect controller is located, in this ca
se, the local IP of the three nodes where the controller is deployed
  ic-nb-port: "6645" # Cluster Interconnect Controller nb por
t, default 6645
  ic-sb-port: "6646" # Cluster Interconnect Controller sb por
t, default 6646
EOF
```

## Join the cluster interconnect

Add a cluster whose network mode is Kube-OVN to the cluster interconnect.

### Prerequisites

The **created subnets**, **ovn-default**, and **join subnets** in a cluster do not conflict with any cluster segment in the cluster interconnection group.

### Procedure of operation

1. In the left navigation bar, click **Clusters > Cluster of clusters**.
2. Click the name of the **cluster** to be added to the cluster interconnect.
3. In the upper right corner, click **Options > Cluster Interconnect**.
4. Click **Join the cluster interconnect**.
5. Select a gateway node for the cluster.
6. Click **Join**.

## Relevant operations

## Update the gateway node information of the interconnected cluster

Update information about cluster gateway nodes that have joined a cluster interconnect group.

### Procedure of operation

1. In the left navigation bar, click **Clusters** > **Cluster of clusters**.
2. Click **Cluster name** for the gateway node information to be updated.
3. In the upper-right corner, click **Operations** > **Cluster Interconnect**.
4. Click **Update Gateway Node** for the cluster whose gateway node information you want to update.
5. Reselect the gateway node for the cluster.
6. Click **Update**.

## Exit cluster interconnection

A cluster that has joined a cluster interconnection group exits cluster interconnection, and when it does, it disconnects the cluster Pod from the external cluster Pod.

### Procedure of operation

1. In the left navigation bar, click **Clusters** > **Cluster of clusters**.
2. Click the name of the **cluster** that you want to decommission.
3. In the upper-right corner, click **Options** > **Cluster Interconnect**.
4. Click **Exit cluster interconnection** for the cluster you want to exit.
5. Enter the cluster name correctly.
6. Click **Exit**.

## Cleaning up Interconnected Cluster Residue

When a cluster is deleted without leaving the interconnected cluster, some residual data may remain on the controller. When you attempt to use these nodes to create a cluster again and

join the interconnected cluster, failures may occur. You can check the detailed error information in the `/var/log/ovn/ovn-ic.log` log of the controller (kube-ovn-controller). Some error messages may include:

```
transaction error: {"details":"Transaction causes multiple rows in xxxxx  
x"}
```

## Operational Steps

1. [Exit the interconnected cluster](#) for the cluster to be joined.
2. Execute the cleanup script in the container or pod.

You can execute the cleanup script directly in either the `ovn-ic-db` container or the `ovn-ic-controller` pod. Choose one of the following methods:

### Method 1: Execute in `ovn-ic-db` container

- Enter the `ovn-ic-db` container and perform the cleanup operation with the following commands.

```
ctr -n k8s.io task exec -t --exec-id ovn-ic-db ovn-ic-db /bin/bash
```

Then execute one of the following cleanup commands:

- Execute the cleanup operation with the name of the original cluster. Replace `<cluster-name>` with the **name of the original cluster**:

```
./clean-ic-az-db.sh <cluster-name>
```

- Execute the cleanup operation with the name of any node in the original cluster. Replace `<node-name>` with the **name of any node in the original cluster**:

```
./clean-ic-az-db.sh <node-name>
```

### Method 2: Execute in `ovn-ic-controller` pod

- Enter the `ovn-ic-controller` pod and perform the cleanup operation with the following commands.

```
kubectl -n kube-system exec -ti $(kubectl get pods -n kube-system -l app=ovn-ic-controller -o custom-columns=NAME:.metadata.name --no-headers) -- /bin/bash
```

Then execute one of the following cleanup commands:

- Execute the cleanup operation with the name of the original cluster. Replace *<cluster-name>* with the **name of the original cluster**:

```
./clean-ic-az-db.sh <cluster-name>
```

- Execute the cleanup operation with the name of any node in the original cluster. Replace *<node-name>* with the **name of any node in the original cluster**:

```
./clean-ic-az-db.sh <node-name>
```

## Uninstalling the Interconnected Cluster

**Note:** [Step 1](#) to [Step 3](#) need to be performed on all **business clusters that have joined the interconnected cluster**.

### Operational Steps

1. Exit the interconnected cluster. There are two specific exit methods, choose one according to your needs.

- Delete the ConfigMap named `ovn-ic-config` in the business cluster. Use the following command.

```
kubectl -n kube-system delete cm ovn-ic-config
```

- Exit the interconnected cluster through [platform operations](#).

2. Enter the Leader Pod of `ovn-central` with the following command.

```
kubectl -n kube-system exec -ti $(kubectl get pods -n kube-system -lovn
-nb-leader=true -o custom-columns=NAME:.metadata.name --no-headers) --
/bin/bash
```

3. Clean up the ts logical switch with the following command.

```
ovn-nbctl ls-del ts
```

4. Log in to the node where the controller is deployed and delete the controller.

- Podman command:

```
podman stop ovn-ic-db
podman rm ovn-ic-db
```

- Containerd command:

```
ctr -n k8s.io task kill ovn-ic-db
ctr -n k8s.io containers rm ovn-ic-db
```

5. Delete the ConfigMap named ovn-ic in the global cluster with the following command.

```
kubectl delete cm ovn-ic -n cpaas-system
```

## Configure Cluster Gateway High Availability

To configure the cluster gateway to be highly available after joining the cluster interconnection, you can perform the following steps:

1. Log in to the cluster that needs to be transformed into a High Availability Gateway and execute the following command to change the `enable-ic` field to `false`.

**Note:** Changing the `enable-ic` field to `false` will disrupt the cluster interconnect until it is set to `true` again.

```
kubectl edit cm ovn-ic-config -n kube-system
```

2. Modify the gateway node configuration by updating the `gw-nodes` field and separating the gateway nodes with English commas; also change the `enable-ic` field to `true`.

```
kubectl edit cm ovn-ic-config -n kube-system

# Configuration example
apiVersion: v1
data:
  auto-route: "true"
  az-name: az1
  enable-ic: "true"
  gw-nodes: 192.168.188.234,192.168.189.54
  ic-db-host: 192.168.178.97
  ic-nb-port: "6645"
  ic-sb-port: "6646"
kind: ConfigMap
metadata:
  creationTimestamp: "2023-06-13T08:01:16Z"
  name: ovn-ic-config
  namespace: kube-system
  resourceVersion: "99671"
  uid: 6163790a-ad9d-4d07-ba82-195b11244983
```

3. Go to the Pod in cluster ovn-central and execute the `ovn-nbctl lrp-get-gateway-chassis {current cluster name}-ts` command to verify that the configuration is in effect.

```
ovn-nbctl lrp-get-gateway-chassis az1-ts

# Return to the display example. In this case, the values of 100 and 99
# are the priority, and the larger the value, the higher the priority of
# the corresponding gateway node to be used.
az1-ts-71292a21-131d-492a-9f0c-0611af458950 100
az1-ts-1de7ee15-f372-4ab9-8c85-e54d61ea18f1 99
```

# Configure Egress Gateway

---

## TOC

### About Egress Gateway

Implementation Details

Notes

Usage

Creating a Network Attachment Definition

Creating a VPC Egress Gateway

Enabling BFD-based High Availability

Configuration Parameters

VPC BFD Port

VPC Egress Gateway

Additional resources

---

## About Egress Gateway

Egress Gateway is used to control external network access for Pods with a group of static addresses and has the following features:

- Achieves Active-Active high availability through ECMP, enabling horizontal throughput scaling
- Implements fast failover (<1s) via BFD
- Supports IPv6 and dual-stack
- Enables granular routing control through NamespaceSelector and PodSelector
- Allows flexible scheduling of Egress Gateway through NodeSelector

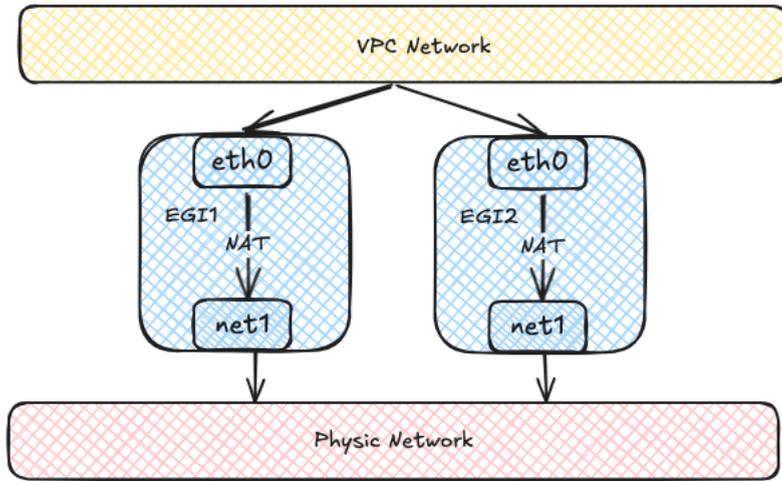
At the same time, Egress Gateway has the following limitations:

- Uses macvlan for underlying network connectivity, requiring Underlay support from the underlying network
- In multi-instance Gateway mode, multiple Egress IPs are required
- Currently, only supports SNAT; EIP and DNAT are not supported
- Currently, recording source address translation relationships is not supported

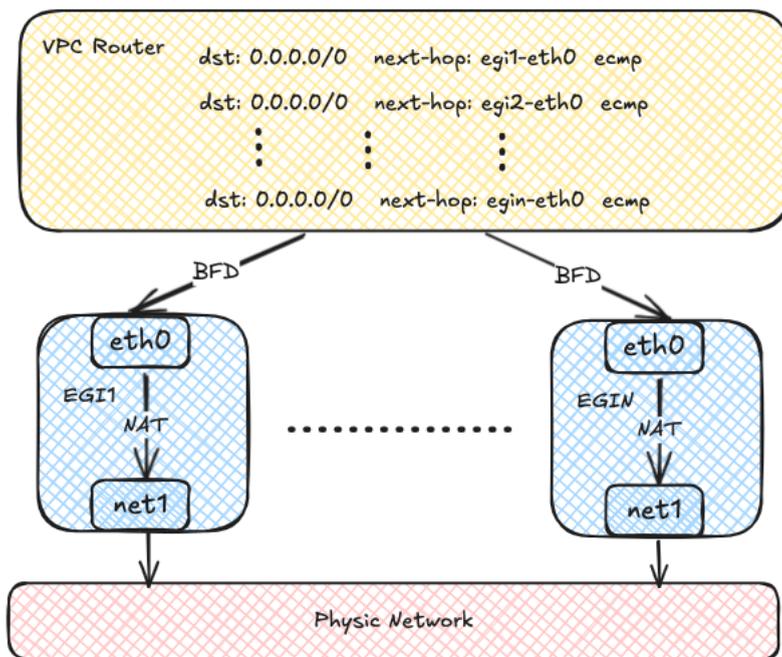
## Implementation Details

Each Egress Gateway consists of multiple Pods with multiple network interfaces. Each Pod has two network interfaces: one joins the virtual network for communication within the VPC, and the other connects to the underlying physical network via Macvlan for external network communication. Virtual network traffic ultimately accesses the external network through NAT within the Egress Gateway instances.

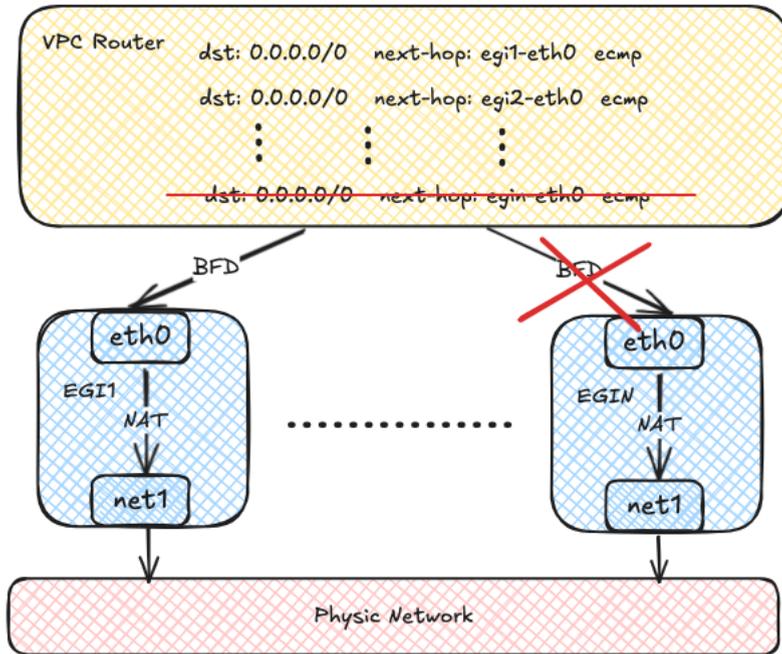
---



Each Egress Gateway instance registers its address in the OVN routing table. When a Pod within the VPC needs to access the external network, OVN uses source address hashing to forward traffic to multiple Egress Gateway instance addresses, achieving load balancing. As the number of Egress Gateway instances increases, throughput can also scale horizontally.



OVN uses the BFD protocol to probe multiple Egress Gateway instances. When an Egress Gateway instance fails, OVN marks the corresponding route as unavailable, enabling rapid failure detection and recovery.



## Notes

*Alauda Container Platform Networking for Multus* **MUST** be installed on the cluster before using Egress Gateway.

To install *Alauda Container Platform Networking for Multus*, refer to [Installing Multus CNI](#).

## Usage

### Creating a Network Attachment Definition

Egress Gateway uses multiple NICs to access both the internal network and the external network, so you need to create a Network Attachment Definition to connect to the external network. An example of using the macvlan plugin with IPAM provided by Kube-OVN is shown below:

```

apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  name: eth1
  namespace: default
spec:
  config: '{
    "cniVersion": "0.3.0",
    "type": "macvlan",
    "master": "eth1", ①
    "mode": "bridge",
    "ipam": {
      "type": "kube-ovn",
      "server_socket": "/run/openvswitch/kube-ovn-daemon.sock",
      "provider": "eth1.default" ②
    }
  }'
---
apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
  name: macvlan1
spec:
  protocol: IPv4
  provider: eth1.default ③
  cidrBlock: 172.17.0.0/16
  gateway: 172.17.0.1
  excludeIps:
    - 172.17.0.2..172.17.0.10

```

- ① Host interface that connects to the external network.
- ② Provider name with a format of `<network attachment definition name>.<namespace>`.
- ③ Provider name used to identify the external network and MUST be consistent with the one in the NetworkAttachmentDefinition.

**TIP**

You can create a Network Attachment Definition with any CNI plugin to access the corresponding network.

## Creating a VPC Egress Gateway

Create a VPC Egress Gateway resource as shown in the example below:

```

apiVersion: kubeovn.io/v1
kind: VpcEgressGateway
metadata:
  name: gateway1
  namespace: default ①
spec:
  replicas: 1 ②
  externalSubnet: macvlan1 ③
  nodeSelector: ④
    - matchExpressions:
      - key: kubernetes.io/hostname
        operator: In
        values:
          - kube-ovn-worker
          - kube-ovn-worker2
  tolerations: ⑤
    - key: node-role.kubernetes.io/control-plane
      operator: Exists
      effect: NoSchedule
  selectors: ⑥
    - namespaceSelector:
        matchLabels:
          kubernetes.io/metadata.name: default
  policies: ⑦
    - snat: true ⑧
      subnets: ⑨
        - subnet1
    - snat: false
      ipBlocks: ⑩
        - 10.18.0.0/16

```

- ① Namespace where the VPC Egress Gateway instances is created.
- ② Replicas of the VPC Egress Gateway instances.
- ③ External subnet that connects to the external network.
- ④ Node selectors used for scheduling the VPC Egress Gateway instances.
- ⑤ Tolerations used for scheduling the VPC Egress Gateway instances.
- ⑥ Namespace selectors and Pod selectors used to select Pods that access the external network via the VPC Egress Gateway.
- ⑦ Policies for the VPC Egress Gateway, including SNAT and subnets/ipBlocks to be applied.
- ⑧ Whether to enable SNAT for the policy.
- ⑨ Subnets to which the policy applies.
- ⑩ IP blocks to which the policy applies.

The above resource creates a VPC Egress Gateway named *gateway1* under the default namespace, and the following Pods will access the external network via the *macvlan1* subnet:

- Pods in the default namespace
- Pods under the *subnet1* subnet
- Pods with IPs in the CIDR *10.18.0.0/16*

#### NOTE

Pods matching *.spec.selectors* will access the external network with SNAT enabled.

After the creation is complete, check out the VPC Egress Gateway resource:

```
$ kubectl get veg gateway1
NAME      VPC      REPLICAS  BFD ENABLED  EXTERNAL SUBNET  PHASE      READY  AGE
gateway1  ovn-cluster  1         false        macvlan1         Completed  true   13s
```

To view more information:

```
kubectl get veg gateway1 -o wide
NAME      VPC      REPLICAS  BFD ENABLED  EXTERNAL SUBNET  PHASE      READY  INTERNAL IPS  EXTERNAL IP
S        WORKING NODES      AGE
gateway1  ovn-cluster  1         false        macvlan1         Completed  true   ["10.16.0.12"] ["172.17.0.11"]
["kube-ovn-worker"] 82s
```

To view the workload:

```
$ kubectl get deployment -l ovn.kubernetes.io/vpc-egress-gateway=gateway1
NAME      READY  UP-TO-DATE  AVAILABLE  AGE
gateway1  1/1    1           1          4m40s

$ kubectl get pod -l ovn.kubernetes.io/vpc-egress-gateway=gateway1 -o wide
NAME      READY  STATUS    RESTARTS  AGE  IP           NODE           NOMINATED NODE  READINESS GATES
gateway1-b9f8b4448-76lhm  1/1    Running   0         4m48s  10.16.0.12  kube-ovn-worker  <none>          <none>
```

To view IP addresses, routes, and iptables rules in the Pod:

```

$ kubectl exec gateway1-b9f8b4448-76lhm -c gateway -- ip address show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: net1@if13: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether 62:d8:71:90:7b:86 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 172.17.0.11/16 brd 172.17.255.255 scope global net1
        valid_lft forever preferred_lft forever
    inet6 fe80::60d8:71ff:fe90:7b86/64 scope link
        valid_lft forever preferred_lft forever
17: eth0@if18: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1400 qdisc noqueue state UP group default
    link/ether 36:7c:6b:c7:82:6b brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.16.0.12/16 brd 10.16.255.255 scope global eth0
        valid_lft forever preferred_lft forever
    inet6 fe80::347c:6bff:fec7:826b/64 scope link
        valid_lft forever preferred_lft forever

$ kubectl exec gateway1-b9f8b4448-76lhm -c gateway -- ip rule show
0:      from all lookup local
1001:   from all iif eth0 lookup default
1002:   from all iif net1 lookup 1000
1003:   from 10.16.0.12 iif lo lookup 1000
1004:   from 172.17.0.11 iif lo lookup default
32766:  from all lookup main
32767:  from all lookup default

$ kubectl exec gateway1-b9f8b4448-76lhm -c gateway -- ip route show
default via 172.17.0.1 dev net1
10.16.0.0/16 dev eth0 proto kernel scope link src 10.16.0.12
10.17.0.0/16 via 10.16.0.1 dev eth0
10.18.0.0/16 via 10.16.0.1 dev eth0
172.17.0.0/16 dev net1 proto kernel scope link src 172.17.0.11

$ kubectl exec gateway1-b9f8b4448-76lhm -c gateway -- ip route show table 1000
default via 10.16.0.1 dev eth0

$ kubectl exec gateway1-b9f8b4448-76lhm -c gateway -- iptables -t nat -S
-P PREROUTING ACCEPT
-P INPUT ACCEPT
-P OUTPUT ACCEPT
-P POSTROUTING ACCEPT
-N VEG-MASQUERADE
-A PREROUTING -i eth0 -j MARK --set-xmark 0x4000/0x4000
-A POSTROUTING -d 10.18.0.0/16 -j RETURN
-A POSTROUTING -s 10.18.0.0/16 -j RETURN
-A POSTROUTING -j VEG-MASQUERADE
-A VEG-MASQUERADE -j MARK --set-xmark 0x0/0xffffffff
-A VEG-MASQUERADE -j MASQUERADE --random-fully

```

Capture packets in the Gateway Pod to verify network traffic:

```

$ kubectl exec -ti gateway1-b9f8b4448-76lhm -c gateway -- bash
nobody@gateway1-b9f8b4448-76lhm:/kube-ovn$ tcpdump -i any -nnve icmp and host 172.17.0.1
tcpdump: data link type LINUX_SLL2
tcpdump: listening on any, link-type LINUX_SLL2 (Linux cooked v2), snapshot length 262144 bytes
06:50:58.936528 eth0 In ifindex 17 92:26:b8:9e:f2:1c ethertype IPv4 (0x0800), length 104: (tos 0x0, ttl 63, id 30481, offset 0, flags [DF], proto ICMP (1), length 84)
    10.17.0.9 > 172.17.0.1: ICMP echo request, id 37989, seq 0, length 64
06:50:58.936574 net1 Out ifindex 2 62:d8:71:90:7b:86 ethertype IPv4 (0x0800), length 104: (tos 0x0, ttl 62, id 30481, offset 0, flags [DF], proto ICMP (1), length 84)
    172.17.0.11 > 172.17.0.1: ICMP echo request, id 39449, seq 0, length 64
06:50:58.936613 net1 In ifindex 2 02:42:39:79:7f:08 ethertype IPv4 (0x0800), length 104: (tos 0x0, ttl 64, id 26701, offset 0, flags [none], proto ICMP (1), length 84)
    172.17.0.1 > 172.17.0.11: ICMP echo reply, id 39449, seq 0, length 64
06:50:58.936621 eth0 Out ifindex 17 36:7c:6b:c7:82:6b ethertype IPv4 (0x0800), length 104: (tos 0x0, ttl 63, id 26701, offset 0, flags [none], proto ICMP (1), length 84)
    172.17.0.1 > 10.17.0.9: ICMP echo reply, id 37989, seq 0, length 64

```

Routing policies are automatically created on the OVN Logical Router:

```

$ kubectl ko nbctl lr-policy-list ovn-cluster
Routing Policies
 31000                ip4.dst == 10.16.0.0/16  allow
 31000                ip4.dst == 10.17.0.0/16  allow
 31000                ip4.dst == 100.64.0.0/16  allow
 30000                ip4.dst == 172.18.0.2  reroute 100.64.0.4
 30000                ip4.dst == 172.18.0.3  reroute 100.64.0.3
 30000                ip4.dst == 172.18.0.4  reroute 100.64.0.2
 29100                ip4.src == $VEG.8ca38ae7da18.ipv4  reroute 10.16.0.12 ①
 29100                ip4.src == $VEG.8ca38ae7da18_ip4  reroute 10.16.0.12 ②
 29000 ip4.src == $ovn.default.kube.ovn.control.plane_ip4  reroute 100.64.0.3
 29000    ip4.src == $ovn.default.kube.ovn.worker2_ip4  reroute 100.64.0.2
 29000    ip4.src == $ovn.default.kube.ovn.worker_ip4  reroute 100.64.0.4
 29000 ip4.src == $subnet1.kube.ovn.control.plane_ip4  reroute 100.64.0.3
 29000    ip4.src == $subnet1.kube.ovn.worker2_ip4  reroute 100.64.0.2
 29000    ip4.src == $subnet1.kube.ovn.worker_ip4  reroute 100.64.0.4

```

- ① Logical Router Policy used by the VPC Egress Gateway to forward traffic from the Pods specified by *.spec.policies*.
- ② Logical Router Policy used by the VPC Egress Gateway to forward traffic from the Pods specified by *.spec.selectors*.

If you need to enable load balancing, modify *.spec.replicas* as shown in the following example:

```

$ kubectl scale veg gateway1 --replicas=2
vpcegressgateway.kubeovn.io/gateway1 scaled

$ kubectl get veg gateway1
NAME      VPC          REPLICAS  BFD ENABLED  EXTERNAL SUBNET  PHASE      READY  AGE
gateway1  ovn-cluster  2         false        macvlan          Completed  true   39m

$ kubectl get pod -l ovn.kubernetes.io/vpc-egress-gateway=gateway1 -o wide
NAME                                READY  STATUS    RESTARTS  AGE  IP             NODE                NOMINATED NODE  READINESS GATES
gateway1-b9f8b4448-76lhm            1/1    Running   0          40m  10.16.0.12    kube-ovn-worker     <none>           <none>
gateway1-b9f8b4448-zd4dl            1/1    Running   0          64s  10.16.0.13    kube-ovn-worker2    <none>           <none>

$ kubectl ko nbctl lr-policy-list ovn-cluster
Routing Policies
 31000                                ip4.dst == 10.16.0.0/16    allow
 31000                                ip4.dst == 10.17.0.0/16    allow
 31000                                ip4.dst == 100.64.0.0/16   allow
 30000                                ip4.dst == 172.18.0.2     reroute 100.64.0.4
 30000                                ip4.dst == 172.18.0.3     reroute 100.64.0.3
 30000                                ip4.dst == 172.18.0.4     reroute 100.64.0.2
 29100                                ip4.src == $VEG.8ca38ae7da18.ipv4  reroute 10.16.0.12, 10.16.0.13
 29100                                ip4.src == $VEG.8ca38ae7da18_ip4   reroute 10.16.0.12, 10.16.0.13
 29000 ip4.src == $ovn.default.kube.ovn.control.plane_ip4  reroute 100.64.0.3
 29000     ip4.src == $ovn.default.kube.ovn.worker2_ip4   reroute 100.64.0.2
 29000     ip4.src == $ovn.default.kube.ovn.worker_ip4       reroute 100.64.0.4
 29000     ip4.src == $subnet1.kube.ovn.control.plane_ip4     reroute 100.64.0.3
 29000     ip4.src == $subnet1.kube.ovn.worker2_ip4           reroute 100.64.0.2
 29000     ip4.src == $subnet1.kube.ovn.worker_ip4             reroute 100.64.0.4

```

## Enabling BFD-based High Availability

BFD-based high availability relies on the VPC BFD LRP function, so you need to modify the VPC resource to enable BFD Port. Here is an example to enable BFD Port for the default VPC:

```

apiVersion: kubeovn.io/v1
kind: Vpc
metadata:
  name: ovn-cluster
spec:
  bfdPort:
    enabled: true ①
    ip: 10.255.255.255 ②
    nodeSelector: ③
      matchLabels:
        kubernetes.io/os: linux

```

- ① Whether to enable the BFD Port.
- ② IP address of the BFD Port, which MUST be a valid IP address that does not conflict with ANY other IPs/Subnets.
- ③ Node selector used to select the nodes where the BFD Port is running in Active-Backup mode.

After the BFD Port is enabled, an LRP dedicated to BFD is automatically created on the corresponding OVN Logical Router:

```
$ kubectl ko nbctl show ovn-cluster
router 0c1d1e8f-4c86-4d96-88b2-c4171c7ff824 (ovn-cluster)
  port bfd@ovn-cluster ①
    mac: "8e:51:4b:16:3c:90"
    networks: ["10.255.255.255"]
  port ovn-cluster-join
    mac: "d2:21:17:71:77:70"
    networks: ["100.64.0.1/16"]
  port ovn-cluster-ovn-default
    mac: "d6:a3:f5:31:cd:89"
    networks: ["10.16.0.1/16"]
  port ovn-cluster-subnet1
    mac: "4a:09:aa:96:bb:f5"
    networks: ["10.17.0.1/16"]
```

- ① BFD Port created on the OVN Logical Router.

After that, set `.spec.bfd.enabled` to `true` in VPC Egress Gateway. An example is shown below:

```
apiVersion: kubeovn.io/v1
kind: VpcEgressGateway
metadata:
  name: gateway2
  namespace: default
spec:
  vpc: ovn-cluster ①
  replicas: 2
  internalSubnet: ovn-default ②
  externalSubnet: macvlan1 ③
  bfd:
    enabled: true ④
    minRX: 100 ⑤
    minTX: 100 ⑥
    multiplier: 5 ⑦
  policies:
    - snat: true
      ipBlocks:
        - 10.18.0.0/16
```

- ① VPC to which the Egress Gateway belongs.
- ② Internal subnet to which the Egress Gateway instances are connected.
- ③ External subnet to which the Egress Gateway instances are connected.
- ④ Whether to enable BFD for the Egress Gateway.
- ⑤ Minimum receive interval for BFD, in milliseconds.
- ⑥ Minimum transmit interval for BFD, in milliseconds.
- ⑦ Multiplier for BFD, which determines the number of missed packets before declaring a failure.

To view VPC Egress Gateway information:

```

$ kubectl get veg gateway2 -o wide
NAME          VPC    REPLICAS  BFD ENABLED  EXTERNAL SUBNET  PHASE      READY  INTERNAL IPS          EXT
ERNAL IPS          WORKING NODES          AGE
gateway2     vpc1   2         true         macvlan          Completed  true   ["10.16.0.102", "10.16.0.103"] ["1
72.17.0.13", "172.17.0.14"] ["kube-ovn-worker", "kube-ovn-worker2"] 58s

$ kubectl get pod -l ovn.kubernetes.io/vpc-egress-gateway=gateway2 -o wide
NAME          READY  STATUS    RESTARTS  AGE    IP           NODE           NOMINATED NODE  RE
ADINESS GATES
gateway2-fcc6b8b87-8lgvx  1/1    Running   0         2m18s  10.16.0.103  kube-ovn-worker2  <none>          <n
one>
gateway2-fcc6b8b87-wmww6  1/1    Running   0         2m18s  10.16.0.102  kube-ovn-worker   <none>          <n
one>

$ kubectl ko nbctl lr-policy-list ovn-cluster
Routing Policies
 31000                ip4.dst == 10.16.0.0/16    allow
 31000                ip4.dst == 10.17.0.0/16    allow
 31000                ip4.dst == 100.64.0.0/16   allow
 30000                ip4.dst == 172.18.0.2     reroute 100.64.0.4
 30000                ip4.dst == 172.18.0.3     reroute 100.64.0.3
 30000                ip4.dst == 172.18.0.4     reroute 100.64.0.2
 29100                ip4.src == $VEG.8ca38ae7da18.ipv4  reroute 10.16.0.102, 10.16.0.103  bfd
 29100                ip4.src == $VEG.8ca38ae7da18_ip4  reroute 10.16.0.102, 10.16.0.103  bfd
 29090                ip4.src == $VEG.8ca38ae7da18.ipv4   drop
 29090                ip4.src == $VEG.8ca38ae7da18_ip4   drop
 29000 ip4.src == $ovn.default.kube.ovn.control.plane_ip4  reroute 100.64.0.3
 29000    ip4.src == $ovn.default.kube.ovn.worker2_ip4  reroute 100.64.0.2
 29000    ip4.src == $ovn.default.kube.ovn.worker_ip4  reroute 100.64.0.4
 29000    ip4.src == $subnet1.kube.ovn.control.plane_ip4  reroute 100.64.0.3
 29000    ip4.src == $subnet1.kube.ovn.worker2_ip4  reroute 100.64.0.2
 29000    ip4.src == $subnet1.kube.ovn.worker_ip4  reroute 100.64.0.4

$ kubectl ko nbctl list bfd
_uuid          : 223ede10-9169-4c7d-9524-a546e24bfab5
detect_mult    : 5
dst_ip         : "10.16.0.102"
external_ids   : {af="4", vendor=kube-ovn, vpc-egress-gateway="default/gateway2"}
logical_port   : "bfd@ovn-cluster"
min_rx        : 100
min_tx        : 100
options        : {}
status        : up

_uuid          : b050c75e-2462-470b-b89c-7bd38889b758
detect_mult    : 5
dst_ip         : "10.16.0.103"
external_ids   : {af="4", vendor=kube-ovn, vpc-egress-gateway="default/gateway2"}
logical_port   : "bfd@ovn-cluster"
min_rx        : 100
min_tx        : 100
options        : {}
status        : up

```

To view BFD connections:

```
$ kubectl exec gateway2-fcc6b8b87-8lgvx -c bfd -- bfd-control status
There are 1 sessions:
Session 1
id=1 local=10.16.0.103 (p) remote=10.255.255.255 state=Up

$ kubectl exec gateway2-fcc6b8b87-wmww6 -c bfd -- bfd-control status
There are 1 sessions:
Session 1
id=1 local=10.16.0.102 (p) remote=10.255.255.255 state=Up
```

**NOTE**

If all the gateway instances are down, egress traffic to which the VPC Egress Gateway is applied will be dropped.

## Configuration Parameters

### VPC BFD Port

Fields	Type	Optional	Default Value	Description	Examples
enabled	boolean	Yes	false	Whether to enable the BFD Port.	true
ip	string	No	-	The IP address used by the BFD Port. Must NOT conflict with other addresses. IPv4, IPv6 and dual-stack are supported.	169.255.255.255
					fdff::1
					169.255.255.255,fdff::1
nodeSelector	matchLabels	Yes	-	Label selectors used to select nodes that carries the BFD Port work. The BFD Port binds an OVN HA Chassis Group of selected nodes and works in Active/Backup mode.	A map of {key,value} pairs. -
	matchExpressions	Yes	-	If this field is not specified, Kube-OVN automatically selects up to three nodes. You can view all OVN HA Chassis Group resources by executing <code>kubectl ko nbctl list ha_chassis_group</code> .	A list of label selector requirements. The requirements are ANDed. -

### VPC Egress Gateway

Fields	Type	Optional	Default Value	Description	
vpc	string	Yes	Name of the default VPC (ovn-cluster)	VPC name.	
replicas	integer/int32	Yes	1	Replicas.	
prefix	string	Yes	-	Immutable prefix of the workload deployment.	
image	string	Yes	-	The image used by the workload deployment.	
internalSubnet	string	Yes	Name of the default subnet within the VPC.	Name of the subnet used to access the internal network.	
externalSubnet		No	-		
internalIPs	string array	Yes	-	IP addresses used for accessing the internal network. IPv4, IPv6 and dual-stack are supported. The number of IPs specified must NOT be less than 1. It is recommended to set the number to <n> to avoid extreme cases where the Pod is not able to access the internal network.	
externalIPs	string array	Yes	-	IP addresses used for accessing the external network. IPv4, IPv6 and dual-stack are supported. The number of IPs specified must NOT be less than 1. It is recommended to set the number to <n> to avoid extreme cases where the Pod is not able to access the external network.	
bfd	enabled	boolean	Yes	false	Whether to enable BFD.
	minRX	integer/int32	Yes	1000	BFD minRX/rx milliseconds.
	minTX	integer/int32	Yes	1000	BFD minTX/tx milliseconds.
	multiplier	integer/int32	Yes	3	BFD multiplier.
policies	snat	boolean	Yes	false	Whether to enable SNAT/MASQ.
	ipBlocks	string array	Yes	-	IP range segments for the egress gateway. Both IPv4 and IPv6 are supported.
	subnets	string array	Yes	-	The VPC subnets for the gateway. Both IPv4 and IPv6 subnets are supported.

Fields	Type	Optional	Default Value	Description		
selectors	matchLabels	object	Yes	-	Namespace selector. An empty label selector matches all namespaces.	
	namespaceSelector					
	matchExpressions	object array	Yes	-	Configure Egress policies by namespace selectors and Pod selectors. SNAT/MASQUERADE will be applied to the matched Pods.	
	podSelector					
	matchLabels	object	Yes	-	Pod selector. An empty label selector matches all Pods.	
	matchExpressions	object array	Yes	-		
nodeSelector	matchLabels	object	Yes	-	Node selector used to select nodes that carries the workload deployment. The workload (Deployment/Pod) will run on the selected nodes.	
	matchExpressions	object array	Yes	-		A list of label requirements are ANDed.
	matchFields	object array	Yes	-		A list of field requirements are ANDed.
tolerations	key	string	Yes	-	Tolerations used for scheduling the VPC Egress Gateway instances.	
	operator	string	Yes	Equal		Key is the tail toleration app means match. If the key is empty, it means match all keys.
	value	string	Yes	-	Operator represents relationship to operators. Exists is equal for value, so it tolerates all tail category.	
					Value is the tail toleration means. If the operator	

Fields	Type	Optional	Default Value	Description
				should be em regular string
effect	string	Yes	-	Effect indicat match. Empty taint effects. When specifi are NoSched PreferNoSch NoExecute.
tolerationSeconds	integer	Yes	-	TolerationSec period of time (which must t NoExecute, c ignored) toler By default, it means tolera (do not evict) Zero and neg treated as 0 ( by the system
trafficPolicy	string	Yes	Cluster	<b>Effective only when BFD is enabled.</b> Available values: <i>Cluster/Local</i> . When set to <i>Local</i> , Egress traffic will be re Egress Gateway instance running on the s available. If the instance is down, Egress traffic will b instances.

## Additional resources

- [Egress Gateway - Kube-OVN Document](#)
- [RFC 5880 - Bidirectional Forwarding Detection \(BFD\)](#)

# Configuring Kube-OVN Network to Support Pod Multi-Network Interfaces (Alpha)

By using Multus CNI, you can add multiple network interfaces with different networks to Pods. Use Kube-OVN network's Subnet and IP CRDs for advanced IP management, implementing subnet management, IP reservation, random allocation, fixed allocation, and other features.

## TOC

### Installing Multus CNI

- Deploying the Multus CNI Plugin

- Creating Subnets

- Creating Pod with Multiple Network Interfaces

- Verifying Dual Network Interface Creation

- Additional Features

  - Fixed IP

  - Additional Routes

## Installing Multus CNI

### Deploying the Multus CNI Plugin

1. Go to **Administrator**.

2. In the left navigation bar, click **Marketplace > Cluster Plugins**.
3. In the search bar, type "multus" to find the Multus CNI plugin.
4. Locate the "**Alauda Container Platform Networking for Multus**" plugin in the list.
5. Click the three dots (:) next to the plugin entry and select **Install**.
6. The plugin will be deployed to your cluster. You can monitor the installation status in the **State** column.

#### NOTE

The Multus CNI plugin serves as middleware between other CNI plugins and Kubernetes, enabling Pods to have multiple network interfaces.

## Creating Subnets

Create an attachnet subnet according to the following example: `network-attachment-definition.yml`.

#### NOTE

The provider format in config is `<NAME>.<NAMESPACE>.ovn`, where `<NAME>` and `<NAMESPACE>` are the name and namespace of this NetworkAttachmentDefinition CR respectively.

```
apiVersion: 'k8s.cni.cncf.io/v1'
kind: NetworkAttachmentDefinition
metadata:
  name: attachnet
  namespace: default
spec:
  config: '{
    "cniVersion": "0.3.0",
    "type": "kube-ovn",
    "server_socket": "/run/openvswitch/kube-ovn-daemon.sock",
    "provider": "attachnet.default.ovn"
  }'
```

After creation, apply the resource:

```
kubectl apply -f network-attachment-definition.yml
```

Use the following example to create the Kube-OVN subnet for the second network interface:

```
subnet.yml
```

#### NOTE

- `spec.provider` must be consistent with the provider in NetworkAttachmentDefinition.
- If you need to use an Underlay subnet, set the `spec.vlan` of the subnet to the VLAN CR name you want to use. Configure other subnet parameters as needed.

```
apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
  name: subnet1
spec:
  cidrBlock: 172.170.0.0/16
  provider: attachnet.default.ovn
```

After creation, apply the resource:

```
kubectl apply -f subnet.yml
```

## Creating Pod with Multiple Network Interfaces

Create a pod according to the following example.

#### NOTE

- The `metadata.annotations` must contain a key-value pair `k8s.v1.cni.cncf.io/networks=default/attachnet`, where the value format is

`<NAMESPACE>/<NAME>` , and `<NAMESPACE>` and `<NAME>` are the namespace and name of the NetworkAttachmentDefinition CR respectively.

- If the Pod needs three network interfaces, configure the value of

`k8s.v1.cni.cncf.io/networks` as `default/attachnet, default/attachnet2` .

```
apiVersion: v1
kind: Pod
metadata:
  name: pod1
  annotations:
    k8s.v1.cni.cncf.io/networks: default/attachnet
spec:
  containers:
    - name: web
      image: nginx:latest
      ports:
        - containerPort: 80
```

After the Pod is created successfully, use the command `kubectl exec pod1 -- ip a` to view the Pod's IP addresses.

## Verifying Dual Network Interface Creation

Use the following command to verify that the dual network interfaces have been created successfully:

```
kubectl exec pod1 -- ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen
1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
151: eth0@if152: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1400 qdisc
noqueue state UP
    link/ether a6:3c:d8:ae:83:06 brd ff:ff:ff:ff:ff:ff
    inet 10.3.0.8/16 brd 10.3.255.255 scope global eth0
        valid_lft forever preferred_lft forever
    inet6 fe80::a43c:d8ff:feae:8306/64 scope link
        valid_lft forever preferred_lft forever
153: net1@if154: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1400 qdisc
noqueue state UP
    link/ether 0a:36:08:01:dc:df brd ff:ff:ff:ff:ff:ff
    inet 172.170.0.3/16 brd 172.170.255.255 scope global net1
        valid_lft forever preferred_lft forever
    inet6 fe80::836:8ff:fe01:dcdf/64 scope link
        valid_lft forever preferred_lft forever
```

## Additional Features

### Fixed IP

- **Primary Network Interface (First Interface):** If you need to fix the IP of the primary network interface, the method is the same as using a fixed IP with a single network interface. Add the annotation `ovn.kubernetes.io/ip_address=<IP>` to the Pod.
- **Secondary Network Interface (Second Interface or Other Interfaces):** The basic method is similar to the primary network interface, with the difference that the `ovn` in the Annotation Key is replaced with the corresponding NetworkAttachmentDefinition provider. Example: `attachnet.default.ovn.kubernetes.io/ip_address=172.170.0.101`.

### Additional Routes

Starting from version 1.8.0, Kube-OVN supports configuring additional routes for secondary network interfaces. When using this feature, add the `routers` field to the config in NetworkAttachmentDefinition and fill in the routes you need to configure. Example:

```
apiVersion: 'k8s.cni.cncf.io/v1'
kind: NetworkAttachmentDefinition
metadata:
  name: attachnet
  namespace: default
spec:
  config: '{
    "cniVersion": "0.3.0",
    "type": "kube-ovn",
    "server_socket": "/run/openvswitch/kube-ovn-daemon.sock",
    "provider": "attachnet.default.ovn",
    "routes": [{
      "dst": "19.10.0.0/16"
    }, {
      "dst": "19.20.0.0/16",
      "gw": "19.10.0.1"
    }]
  }'
```

# Configure IPPool

IPPool is a more granular IPAM management unit than Subnet. You can subdivide the subnet segment into multiple units through IPPool, and each unit is bound to one or more namespaces.

## TOC

### Instructions

Create IPPool

Use IPPool

Precautions

## Instructions

### Create IPPool

Below is an example:

```

apiVersion: kubeovn.io/v1
kind: IPPool
metadata:
  name: pool-1
spec:
  subnet: ovn-default ❶
  ips: ❷
  - "10.16.0.201"
  - "10.16.0.210/30"
  - "10.16.0.220..10.16.0.230"
  namespaces: ❸
  - ns-1

```

- ❶ Subnet to which the IP pool belongs.
- ❷ IP ranges. Supported formats: `<IP>`, `<CIDR>` and `<IP1>..<IP2>`. Both IPv4 and IPv6 are supported.
- ❸ Optional namespaces the IP pool is bound to. Pods in a bound namespace will only get IPs from the bound pool(s), not other ranges in the subnet(s).

## Use IPPool

To assign IPs randomly from the IP pool, simply bind the IP pool to the desired namespace(s). When Pods in the bound namespace are created, their IPs will be allocated from the corresponding IP pool(s).

### NOTE

Before binding an IP pool to a namespace, make sure only the subnet to which the IP pool belongs is bound to the namespace.

You can also assign an IP pool to Pods through annotation:

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-1
  annotations:
    ovn.kubernetes.io/ip_pool: pool-1
spec:
  containers:
  - name: web
    image: nginx:latest
```

For workloads, use annotation in the Pod template of the Deployment, StatefulSet, etc.:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: deploy-1
spec:
  replicas: 1
  selector:
    matchLabels:
      app: web
  template:
    metadata:
      labels:
        app: web
      annotations:
        ovn.kubernetes.io/ip_pool: pool-1
    spec:
      containers:
      - name: web
        image: nginx:latest
```

## Precautions

1. To ensure compatibility with function *Fixed Addresses*, name of the IP pool cannot be an IP address.
2. IP addresses out of the subnet range are allowed, while these IPs will not be effective.

3. Different IP pools belonging to the same subnet cannot have overlapping IP ranges.
4. The `.spec.ips` field can be updated whenever necessary. Any changes will take effect immediately.
5. An IP pool will inherit the reserved IP of the subnet. When randomly assigning an IP address from an IP pool, the reserved IP within the IP pool range will be skipped.
6. When randomly assigning an IP address from a subnet, IP ranges of all IP pools in the subnet will be excluded.
7. Multiple IP pools can be bound to the same namespace.

# Configure MTU

In Kube-OVN, the MTU (Maximum Transmission Unit) setting is crucial for ensuring optimal network performance and avoiding packet fragmentation. The MTU defines the maximum size of a packet that can be transmitted over the network.

By default, Kube-OVN detects the MTU of the underlying physical network interface and sets the MTU for the virtual network interfaces accordingly. However, there are scenarios where you might need to customize the MTU settings for your Kube-OVN networking components.

---

## TOC

- [Default MTU Behavior](#)

  - Customizing MTU Settings

    - Global MTU Configuration

    - Per-Subnet MTU Configuration

---

## Default MTU Behavior

Kube-OVN automatically detects the MTU of the physical network interface on the host machine, and sets the MTU for the Pod and OVS interfaces accordingly.

In overlay networks, Kube-OVN reduces the MTU to accommodate the VXLAN or Geneve encapsulation overhead. Here is how the MTU is calculated:

Encapsulation Type	Tunnel IP Version	MTU Calculation
Geneve	IPv4	Physical Network Interface MTU - 100
	IPv6	Physical Network Interface MTU - 120
VXLAN	IPv4	Physical Network Interface MTU - 50
	IPv6	Physical Network Interface MTU - 70

In underlay networks, the MTU is set to match the physical network interface MTU.

## Customizing MTU Settings

MTU settings can be customized globally for overlay networks or on a per-subnet basis.

### WARNING

Adjusting MTU settings incorrectly can lead to network performance issues, including packet loss and fragmentation. Ensure that the MTU settings are compatible with your underlying network infrastructure before making changes.

### Critical Consideration When Increasing MTU

**When increasing MTU from a smaller to a larger value, you must restart all Pods** to ensure the new MTU takes effect uniformly across the cluster.

#### Why is this necessary?

OVS internal ports (such as `ovn0`) automatically adopt the **smallest MTU** among all interfaces connected to the `br-int` bridge as their own MTU. This behavior creates a potential issue in the following scenario:

1. You configure a larger MTU value for new Pods
2. Some existing Pods still use the original smaller MTU
3. The `ovn0` interface retains the smaller MTU due to the minimum MTU rule

4. Traffic from Pods with the larger MTU gets dropped when packets exceed the `ovn0` MTU

**Solution:** After increasing MTU settings, recreate all Pods to ensure consistent MTU configuration across the entire network path.

## NOTE

MTU configuration changes will only take effect for newly created Pods. Existing Pods will retain their original MTU settings until they are recreated. It's recommended to plan for Pod restarts when changing MTU settings.

## Global MTU Configuration

To set a global MTU for overlay networks, you can modify the command parameter of the `kube-ovn-cni` DaemonSet. For example, to set the global MTU to 1400, you would update the DaemonSet as follows:

```
kubectl -n kube-system edit ds kube-ovn-cni
```

Then, add or update the `--mtu=1400` parameter to the container's arguments:

```

apiVersion: apps/v1
kind: DaemonSet
metadata:
  kubernetes.io/description: |
    This daemon set launches the kube-ovn cni daemon.
  name: kube-ovn-cni
  namespace: kube-system
spec:
  selector:
    matchLabels:
      app: kube-ovn-cni
  template:
    metadata:
      labels:
        app: kube-ovn-cni
        component: network
        type: infra
    spec:
      containers:
        - name: cni-server
          args:
            - --mtu=1400

```

**NOTE**

The global MTU setting only affects overlay networks. For underlay networks, the MTU remains the same as the physical network interface MTU by default.

## Per-Subnet MTU Configuration

You can also set the MTU for individual subnets by specifying the `mtu` field in the subnet configuration. For example, to set the MTU for a specific subnet to 1450, you can use the following command:

```
kubectl patch subnet my-subnet --type merge -p '{"spec": {"mtu": 1450}}'
```

This setting will override the global MTU setting for that particular subnet.



---

# Configure Endpoint Health Checker

---

## TOC

| [Overview](#)

Key Features

Installation

    Install via Marketplace

How It Works

    Health Check Mechanism

        Core Functionality

        Health Check Process

        Performance Improvement

How To Activate

    Pod-level annotation (Recommended)

        For ALB

        For IngressNginx

        For EnvoyGateway

        For Custom Deployment

    Pod-level readinessGates (Legacy)

Uninstallation

---

## Overview

---

The Endpoint Health Checker is a cluster plugin designed to monitor and manage the health status of service endpoints in k8s cluster. It automatically removes unhealthy endpoints from service to ensure traffic is only routed to healthy instances, improving overall service reliability and availability.

## Key Features

- **Automatic Health Monitoring:** Continuously monitors the health status of service endpoints in k8s cluster
- **Load Balancer Integration:** Automatically removes unhealthy endpoints from service
- **Service Availability:** Ensures traffic is only directed to healthy, available endpoints
- **Rapid Failover:** Reduces endpoint switching time from 40s to 10s during node power outages

## Installation

### Install via Marketplace

1. Navigate to **Administrator > Marketplace > Cluster Plugins**.
2. Search for "**Alauda Container Platform Endpoint Health Checker**" in the plugin list.
3. Click **Install** to open the installation configuration page.
4. In the deployment configuration dialog, you can optionally configure the following parameters:

Parameter	Description
<b>Node Selectors</b>	Configure label selectors to specify which nodes the Endpoint Health Checker components should run on. Click <b>Add</b> to add multiple label key-value pairs.
<b>Node Tolerations</b>	Configure tolerations to allow Endpoint Health Checker components to be scheduled on nodes with specific taints. Click <b>Add</b> to add multiple tolerations with Key, Value, and Type.

5. Click **Install** to deploy the plugin.
6. Wait for the plugin status to change to "**Ready**".

## How It Works

### Health Check Mechanism

The Endpoint Health Checker is a dedicated health monitoring component that ensures only healthy endpoints receive traffic. It operates by monitoring service endpoints and automatically managing their availability status.

### Core Functionality

The Endpoint Health Checker works by:

1. **Service Discovery:** Identifies services and pods configured for health monitoring in the cluster.
2. **Pod Health Monitoring:** Monitors the readiness and liveness probe status of pods backing the service endpoints
3. **Active Health Checks:** Performs active health assessments using configurable criteria:
  - **TCP connectivity checks:** Establishes TCP connections to verify port accessibility
4. **Endpoint Management:** Automatically removes unhealthy endpoints from service endpoint lists to prevent traffic routing to failed instances

### Health Check Process

The health checking process involves:

- **Probe Integration:** Leverages Kubernetes readiness and liveness probe results as initial health indicators
- **Network Connectivity:** Sends TCP packets to target endpoint ports to verify accessibility
- **Response Validation:** Evaluates response status, timing, and content to determine endpoint health

- **Automatic Failover:** Removes unresponsive or failed endpoints from service endpoint lists

## Performance Improvement

- **Previous Method:** Relied on kubelet heartbeat detection with up to 40 seconds delay
- **Current Method:** Active endpoint health checking with 10 second detection and switching time
- **Improvement:** Significantly improves service availability during node failures in ALB + MetalLB environments

## How To Activate

Health checking can be activated through two methods:

### Pod-level annotation (Recommended)

#### For ALB

set `alb.cpaas.io/pod-annotations` annotation of `ALB2`

```
apiVersion: crd.alauda.io/v2
kind: ALB2
metadata:
  annotations:
    alb.cpaas.io/pod-annotations: '{"endpoint-health-checker.io/enable
d":"true"}'
  name: demo-alb
spec:
  config:
    loadbalancerName: demo-alb
    nodeSelector:
      ingress: 'true'
    replicas: 1
  type: nginx
```

#### For IngressNginx

1. [Install ingress-nginx](#)
2. Set `podAnnotations` in `.spec.controller.podAnnotations` of `IngressNginx`.

```
apiVersion: ingress-nginx.alauda.io/v1
kind: IngressNginx
metadata:
  name: demo
  namespace: ingress-nginx-operator
spec:
  controller:
    replicaCount: 1
    podAnnotations:
      endpoint-health-checker.io/enabled: 'true'
```

## For EnvoyGateway

1. [Install envoy-gateway-operator](#)
2. Set `annotations` in `.spec.provider.kubernetes.envoyDeployment.patch.value.spec.template.metadata.annotations` of `EnvoyProxy`.

```

apiVersion: gateway.networking.k8s.io/v1
kind: Gateway
metadata:
  name: demo
spec:
  infrastructure:
    parametersRef:
      group: gateway.envoyproxy.io
      kind: EnvoyProxy
      name: demo
  gatewayClassName: envoy-gateway-operator-cpaas-default
  listeners:
    - name: http
      port: 80
      protocol: HTTP
---
apiVersion: gateway.envoyproxy.io/v1alpha1
kind: EnvoyProxy
metadata:
  name: demo
spec:
  provider:
    kubernetes:
      envoyDeployment:
        replicas: 1
        patch:
          type: StrategicMerge
          value:
            spec:
              template:
                metadata:
                  annotations:
                    endpoint-health-checker.io/enabled: 'true'
            container:
              imageRepository: registry.alauda.cn:60080/acp/envoyproxy/envoy
          type: Kubernetes

```

## For Custom Deployment

set `annotations` in `.spec.template.metadata.annotations` of `Deployment`

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: demo
spec:
  replicas: 1
  selector:
    matchLabels:
      app: demo
  template:
    metadata:
      labels:
        app: demo
      annotations:
        endpoint-health-checker.io/enabled: 'true'
    spec:
      containers:
        - name: container
          ports:
            - containerPort: 8080
          livenessProbe:
            tcpSocket:
              port: 8080
            initialDelaySeconds: 15
            periodSeconds: 10
          readinessProbe:
            tcpSocket:
              port: 8080
            initialDelaySeconds: 5
            periodSeconds: 5
```

## Pod-level readinessGates (Legacy)

Configure readinessGates in the pod spec for older versions:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: pod-legacy
  namespace: cpaas-system
spec:
  replicas: 3
  selector:
    matchLabels:
      app: pod-legacy
  template:
    metadata:
      labels:
        app: pod-legacy
    spec:
      readinessGates:
        - conditionType: 'endpointHealthCheckSuccess'
      containers:
        - name: container
          image: your-image:latest
          ports:
            - containerPort: 8080
          livenessProbe:
            tcpSocket:
              port: 8080
            initialDelaySeconds: 15
            periodSeconds: 10
          readinessProbe:
            tcpSocket:
              port: 8080
            initialDelaySeconds: 5
            periodSeconds: 5
```

**Note:** The readinessGates configuration is from an older version. It's recommended to use the pod annotation `endpoint-health-checker.io/enabled: 'true'` for new deployments.

## Uninstallation

To uninstall the Endpoint Health Checker:

1. Navigate to **Administrator > Marketplace > Cluster Plugins**.
2. Find the installed "**Endpoint Health Checker**" plugin.
3. Click the options menu and select **Uninstall**.
4. Confirm the uninstallation when prompted.

---

# Tasks for ALB

---

## TOC

[How To Set NodeSelector And Tolerations For alb-operator](#)

How To Set NodeSelector And Tolerations For alb

---

# How To Set NodeSelector And Tolerations For alb-operator

update the `deployment` resources

```
# example of nodeSelector and tolerations
kubectl patch subscription ingress-nginx-operator -n ingress-nginx-operator --type='merge' -p '{
  "spec": {
    "config": {
      "nodeSelector": {
        "node-role.kubernetes.io/infra": ""
      },
      "tolerations": [
        {
          "effect": "NoSchedule",
          "key": "node-role.kubernetes.io/infra",
          "operator": "Equal",
          "value": "reserved"
        }
      ]
    }
  }
}'
```

## How To Set NodeSelector And Tolerations For alb

update the `alb` resources

```
kubectl patch alb2 $NAME -n $MS --type='merge' -p '{
  "metadata": {
    "annotations": {
      "alb.cpaas.io/toleration": "[{\"key\": \"node-role.kubernetes.io/infra\", \"operator\": \"Equal\", \"value\": \"reserved\", \"effect\": \"NoSchedule\"}]"
    }
  },
  "spec": {
    "config": {
      "nodeSelector": {
        "node-role.kubernetes.io/infra": ""
      }
    }
  }
}'
```

# Trouble Shooting

---

[How to Solve Inter-node Comm](#) [Find Who Cause the Error](#)

# How to Solve Inter-node Communication Issues in ARM Environments?

When using lower kernel versions and certain domestic network cards, there may be an issue where the network card computes checksums incorrectly after enabling Checksum Offload. This can lead to communication failures between nodes in the Kube-OVN Overlay network. The specific solutions are as follows:

- **Solution 1: Upgrade the Kernel Version.** It is recommended to upgrade the kernel version to 4.19.90-25.16.v2101 or a higher version.
- **Solution 2: Disable Checksum Offload.** If it is not possible to immediately upgrade the kernel version and inter-node communication issues occur, you can disable the Checksum Offload for the physical network card using the following command.

```
ethtool -K eth0 tx off
```

# Find Who Cause the Error

The `X-ALB-ERR-REASON` field in the response header of the error request will indicate the reason for the error.

The error reason might be:

```
InvalidBalancer : no balancer found for xx # it means no endpoint found f  
or the service
```

```
BackendError : read xxx byte data from backend # it means the backend did  
give response, the error code is not cause by alb.
```

```
InvalidUpstream : no rule match # it means the request does not match any  
rule, so alb return 404.
```