Networking

Overview

Overview

Understanding networking

Core network layers and components

GatewayAPI

Comparison Among Service, Ingress, Gateway API

Network Traffic Flow

Trademark Notice

Networking Operators

Alauda Container Platform Loa

Understanding MetalLB Operator

Algorithm for Selecting VIP Host Nodes

Custom Resource Definitions

Constraints and Limitations

Installing and Uninstalling the MetalLB Op

Alauda Container Platform Ingr

Overview

Installation

Configuration Via Web Console

Configuration Via YAML

Additional Resources

Alauda Con

Overview

Install Envoy G

Next Step

Trademark Noti

Upgrade the MetalLB Operator
Trademark Notice
Trademark Notice

ALB Operat

Network Security

Understanding Network Policy

Introduction
Evaluation of network policy by KUBE-OV
What is the difference between the Admin
namespace scope?

Admin Network Policy

Notes
Creating AdminNetworkPolicy or Baseline
Creating AdminNetworkPolicy or Baseline

Network Po

Creating Netwo
Creating Netwo
Reference

Ingress and Load Balancing

Ingress and Load Balancing with Envoy Gateway

Overview
Network Flow Architecture
Configuring Routes
Configuring Ingress Cluster Traffic
Load Balancing Strategies
TLS and Security
Integration with MetalLB
Troubleshooting

Trademark Notice

Guides

Configure Domain

Example Domain custom resource (CR)
Creating Domain by using the web console
Creating Domain by using the CLI
Use the Same Domain Across Multiple Clu
Subsequent Actions
Additional resources

Creating Certificates

Creating a certificate by using the web cor

Configure S

Why Service is
Example Cluste
Headless Servi

Configure Ingresses

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

Trademark Noti

Configure Subnets

IP Allocation Rules
Calico Network
Kube-OVN Network
Subnet Management

Configure Alauda Container Platform Load Ba

Prerequisites
Configure MetalLB Components
Configure an External IP Pool by using the web console
Configure BGP Peers by using the web console
Configure an External IP Pool with L2Advertisement or BGPA

Configure GatewayAPI Gateway

Overview
Prerequisites
Gateway Basics

Configure G

Overview
Prerequisites

Create Gateway
View Gateway Details
Listener and Route Reference
Next Step
Trademark Notice

Policy Attachme
Policy Attachme
Create Policies
SecurityPolicy
BackendTLSPo

Configure GatewayAPI Route

Configure GatewayAPI ListenerSet

Overview
Prerequisites
Configuration
Attach a Route
Policy behavior
View ListenerSet Details
Next Step
Related Tasks
Trademark Notice

Configure E

Overview
Quota model
Create a quota

Configure ALB

ALB
Frontend

Configure Alauda Container Platform Node DNS Cache

Overview
Key Features
Important Notes
Installation
How It Works
Configuration
Known Risks and Temporary Workaround
Trademark Notice

Configure Alauda Container Platform Cluster

Overview
Configuration
Trademark Notice

Network Observability

Network Observability

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
- Trademark Notice

Tasks for Envoy Gateway

- Overview
- Prerequisites
- Advanced Tasks
- Related Documentation
- More Configuration
- Trademark Notice

Kube OVN

Configure Endpoint Health Checker

- Overview
- Key Features
- Installation
- How It Works
- How To Activate
- Uninstallation
- Trademark Notice

Gateway AP

- Overview
- ACP release ma
- Trademark Noti

Soft Data C

alb

Task: Migra

- Introduction
- Prerequisites

Basic HTTP Ro
Route Timeouts
HTTP Strict Tra
Cookie-Based S
Path-Based Ro
Header Modifica
Connection Lim
Rate Limiting
IP Allowlist/Bloc
URL Rewrite
Cross-Namespa
Default TLS Ce
TLS Re-encrypt
Edge Terminatio
TLS Passthroug
Feature Compa
Migration Strate
Related Docum
Trademark Noti

Trouble Shooting

How to Solve Inter-node Comm **Find Who Cause the Error**

Overview

TOC

- Understanding networking
  - Service Discovery & Load Balancing
  - Traffic Management & Routing
  - Network Security
  - Observability
- Core network layers and components
- GatewayAPI
  - Gateway API for Ingress
  - Gateway API concepts
- Comparison Among Service, Ingress, Gateway API
  - For L4 (TCP/UDP) Traffic
    - LoadBalancer Service (Recommended)
    - Gateway API TCP/UDP Routes
    - Recommendation
  - For L7(HTTP/HTTPS) Traffic
    - Ingress
    - GatewayAPI(Recommended)
- Network Traffic Flow
- Trademark Notice

Understanding networking

In a cloud-native environment, the core role of networking is to act as the central nervous system and circulatory system. It is the foundational enabler for key application characteristics like elasticity, resilience, and observability.

The responsibilities of cloud-native networking can be broken down into several key areas:

Service Discovery & Load Balancing

Responsibility: Automatically discover running microservice instances and intelligently distribute traffic among them.

Implementation:

- **Service Discovery**
  - Pods register with central registry (e.g., `etcd`)
  - Services query registry to find available instances
  - Kubernetes `CoreDNS` provides built-in discovery via service names
- **Load Balancing**
  - Kubernetes `Service` resources get Virtual IP (VIP)
  - Acts as built-in load balancer to backend Pods (Endpoints)
  - Distributes requests evenly across instances

Traffic Management & Routing

Responsibility: Control traffic with fine-grained rules to enable advanced deployment patterns.

Implementation:

- **Ingress**
  - Cluster's "entry gate" for external HTTP/HTTPS traffic

  - Host/path-based routing
  - SSL termination
- **Service Mesh** (Istio, Linkerd)
  - **Canary Releases**: Direct percentage of traffic to new versions
  - **Fault Injection**: Simulate service failures for resilience testing
  - **Timeouts, Retries & Circuit Breaking**: Improve fault tolerance
  - **Traffic Mirroring**: Copy production traffic to testing environments

Network Security

Responsibility: Enforce "Zero Trust" security model for authorized service communication.

Implementation:

- **Network Policies**
  - Firewall-like rules for Pod communication
  - Define which Pods can communicate with each other
  - Example: "Only frontend Pods can access backend database Pods"
- **Mutual TLS (mTLS)**
  - Encrypts and authenticates all service-to-service communication
  - Ensures data security and identity verification
  - Widely used in service meshes

Observability

Responsibility: Provide clear visibility into network behavior so that operators can understand traffic patterns and quickly diagnose connectivity or performance issues.

Implementation:

- **Continuous Visibility**
  - Collect cluster-wide network flow logs with NetObserv

  - Enrich flow records with Kubernetes metadata
  - Store and query data for troubleshooting and auditing
- **On-Demand Troubleshooting**
  - Use the CLI to capture flows or packets for short-term analysis
  - Support real-time inspection in TUI mode or background capture tasks
  - Export results for offline analysis tools such as Wireshark
- **Typical Use Cases**
  - Identify dropped connections, latency hotspots, and unexpected traffic paths
  - Verify service-to-service communication and detect anomalies
  - Investigate incidents with both summarized flows and raw packets

Learn more about Networking Observability.

Core network layers and components

The container network is a comprehensive networking solution designed for cloud-native applications, ensuring seamless east-west communication within clusters and efficient north-south traffic management across external networks, while providing essential networking functionalities. It consists of these core components:

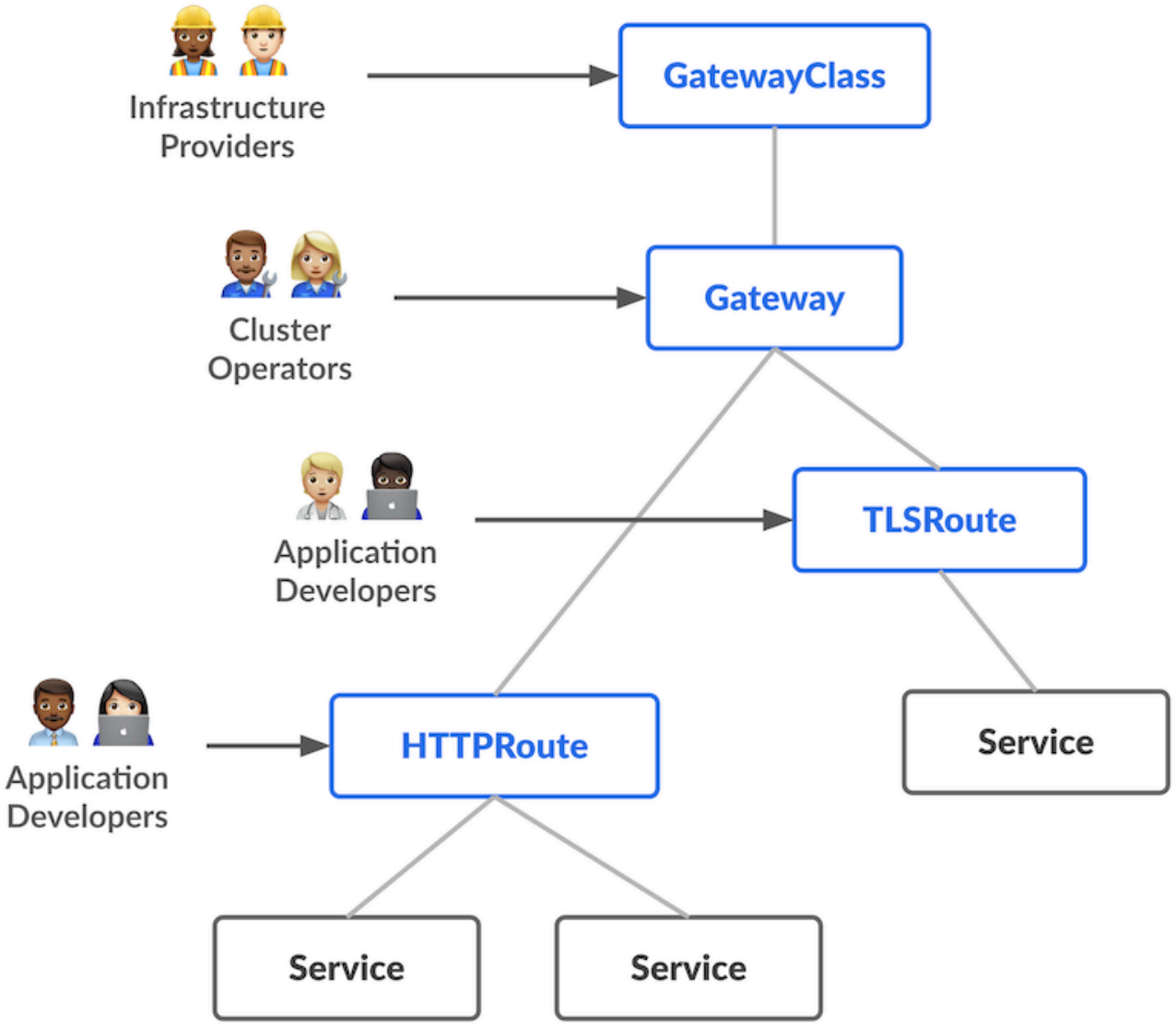
- Container Network Interfaces (CNIs) for east-west traffic management within the cluster.
- Ingress Gateway Controller ALB (Deprecated), NGINX Ingress Controller or Envoy Gateway (Recommended) for managing HTTPS ingress traffic.
- MetalLB for handling LoadBalancer type Services.
- Additionally, it provides robust network security and encryption features to ensure secure communication.

GatewayAPI

Gateway API is an official Kubernetes project focused on L4 and L7 routing in Kubernetes. This project represents the next generation of Kubernetes Ingress, Load Balancing, and

Service Mesh APIs. From the outset, it has been designed to be generic, expressive, and role-oriented.

The overall resource model focuses on 3 separate personas and corresponding resources that they are expected to manage:



Most of the configuration in this API is contained in the Routing layer. These protocol-specific resources (HTTPRoute, GRPCRoute, etc.) enable advanced routing capabilities for both Ingress and Mesh.

Gateway API for Ingress

When using Gateway API to manage ingress traffic, the Gateway resource defines a point of access at which traffic can be routed across multiple contexts -- for example, from outside the cluster to inside the cluster (north/south traffic).

Each Gateway is associated with a GatewayClass, which describes the actual kind of gateway controller that will handle traffic for the Gateway; individual routing resources (such as HTTPRoute) are then associated with the Gateway resources. Separating these different concerns into distinct resources is a critical part of the role-oriented nature of Gateway API, as well as allowing for multiple kinds of gateway controllers (represented by GatewayClass resources),

Gateway API concepts

The following design goals drive the concepts of Gateway API. These demonstrate how Gateway aims to improve upon current standards like Ingress.

- Role-oriented - Gateway is composed of API resources which model organizational roles that use and configure Kubernetes service networking.
- Portable - This isn't an improvement but rather something that should stay the same. Just as Ingress is a universal specification with numerous implementations, Gateway API is designed to be a portable specification supported by many implementations.
- Expressive - Gateway API resources support core functionality for things like header-based matching, traffic weighting, and other capabilities that were only possible in Ingress through custom annotations.
- Extensible - Gateway API allows for custom resources to be linked at various layers of the API. This makes granular customization possible at the appropriate places within the API structure.

Some other notable capabilities include:

- GatewayClasses - GatewayClasses formalize types of load balancing implementations. These classes make it easy and explicit for users to understand what kind of capabilities are available via the Kubernetes resource model.
- Shared Gateways and cross-Namespace support - They allow the sharing of load balancers and VIPs by permitting independent Route resources to attach to the same Gateway. This allows teams (even across Namespaces) to share infrastructure safely without direct coordination.
- Typed Routes and typed backends - Gateway API supports typed Route resources and also different types of backends. This allows the API to be flexible in supporting various

protocols (like HTTP and gRPC) and various backend targets (like Kubernetes Services, storage buckets, or functions).

For more detailed descriptions of the Gateway API, please refer to the gateway-api documentation↗.

Comparison Among Service, Ingress, Gateway API

Alauda Container Platform supports multiple ingress traffic specifications in the Kubernetes ecosystem. Compare Service, Ingress, and Gateway API to choose the traffic entry pattern that fits your workload and operational requirements.

For L4 (TCP/UDP) Traffic

Both LoadBalancer-type Services and Gateway API (TCP/UDP Routes) can expose Layer 4 traffic externally. However, they differ significantly in their implementation approach and performance characteristics.

LoadBalancer Service (Recommended)

Implementation: Kernel-space forwarding

- Traffic forwarding is handled directly by the Linux kernel (iptables/IPVS/eBPF)
- Minimal overhead and near-native performance

Advantages:

- High performance with low latency
- Lower CPU and memory overhead
- Battle-tested and mature technology

Gateway API TCP/UDP Routes

Implementation: User-space proxy

- Implemented by Envoy Gateway (or other gateway controllers)

- Traffic must traverse from kernel space to user space and back
- Additional processing overhead in the application layer

Disadvantages:

- Performance degradation compared to kernel-space solutions
- Higher resource consumption (CPU/memory)
- Additional latency due to user-space context switching

Recommendation

We recommend using LoadBalancer-type Services for L4 traffic routing due to their superior performance and lower resource overhead.

Currently, we support LoadBalancer-type services through MetalLB.

For L7(HTTP/HTTPS) Traffic

While Ingress, GatewayAPI, can all expose L7 traffic externally, they differ in their capabilities.

Ingress

Ingress is the standard specification adopted by the Kubernetes community and are recommended for default use. The Ingress is handled by ALB instances that are managed by the platform administrator.

GatewayAPI(Recommended)

Gateway API is the next-generation routing standard for Kubernetes, designed to address the limitations of Ingress and provide more powerful, flexible, and standardized traffic management capabilities. Compared to Ingress, Gateway API offers the following advantages:

1. **Role-Oriented Design**

   - **Clear separation of concerns**: Infrastructure admin (GatewayClass, Gateway) vs. Application developer (Routes)
   - Ingress mixes infrastructure and application concerns in a single resource

2. **Expressive Routing Capabilities**

   - Rich matching rules: HTTP headers, query parameters, methods, etc.
   - Ingress is limited to host and path matching
   - Built-in support for traffic splitting, mirroring, and advanced policies

3. **Protocol Support**

   - Native support for HTTP, HTTPS, TCP, UDP, gRPC, and TLS passthrough
   - Ingress primarily focuses on HTTP/HTTPS only

4. **Extensibility**

   - Type-safe extension points through CRDs (e.g., HTTPRoute filters, policy attachments)
   - Ingress relies heavily on vendor-specific annotations, leading to portability issues

5. **Cross-Namespace Routing**

   - Routes can reference Services in different namespaces (with ReferenceGrant)
   - Ingress is typically limited to same-namespace references

6. **Multiple Listeners per Gateway**

   - A single Gateway can handle multiple ports, protocols, and hostnames
   - Ingress typically requires one resource per configuration

7. **Portability and Standardization**

   - Vendor-neutral API with conformance tests
   - Reduces lock-in and improves interoperability between implementations
   - Ingress implementations vary widely in capabilities and annotations

8. **Attachment and Selection Model**

   - Routes explicitly attach to Gateway listeners via `parentRefs`
   - Clearer relationships and easier troubleshooting
   - Ingress uses IngressClass which is less flexible

Currently, we support GatewayApi through Envoygateway. For migration from ingress to gatewayapi please refer to migration guide ↗

Network Traffic Flow

Example network traffic flow through an Alauda Container Platform cluster.



Flow explanation:

1. **Client (Browser / curl).**
   The user sends an HTTPS request to `https://www.example.com`. The client works at Layer 7, initiating a DNS lookup.
2. **DNS Resolution.**
   DNS translates the domain name into a public IP address (e.g. `34.23.88.11`).
3. **[L4] External Load Balancer.**
   Operates at Layer 4 (TCP/UDP). It forwards incoming connections to backend nodes in the cluster. Examples: AWS NLB, GCP TCP LB, MetalLB.
4. **[L7] Envoy Gateway / Ingress Controller.**
   Operates at Layer 7 (Application Layer). Handles:
   - TLS termination
   - Hostname and path-based routing
   - Policies and authentication

It routes traffic to the matching Kubernetes Service.

5. **[L4] Kubernetes Service (ClusterIP)**
   Acts as an internal Layer 4 load balancer inside the cluster. Distributes requests to backend Pods based on selectors.
6. **Pod (Application)**
   The final destination where the app runs and processes the request. The response follows the reverse path back to the client.

Trademark Notice

Envoy® is a registered trademark of The Linux Foundation.

NGINX® is a registered trademark of F5, Inc. in the U.S. and in other countries.

MetalLB™ is a trademark of The Linux Foundation.

CoreDNS™ is a trademark of The Linux Foundation.

Alauda is an independent vendor. This product is not affiliated with, endorsed by, or sponsored by The Linux Foundation or F5, Inc. All trademarks are the property of their respective owners and are used here for identification purposes only.

Networking Operators

Alauda Container Platform Load Balancer

Alauda Container Platform Load Balancer

Understanding MetalLB Operator

Algorithm for Selecting VIP Host Nodes

Custom Resource Definitions

Constraints and Limitations

Installing and Uninstalling the MetalLB Operator

Upgrade the MetalLB Operator

Trademark Notice

Alauda Container Platform Ingress Controller

Alauda Container Platform Ingress Controller

Overview

Installation

Configuration Via Web Console

Configuration Via YAML
Additional Resources
Trademark Notice

Alauda Container Platform Gateway API Controller

Alauda Container Platform Gateway API Controller

Overview
Install Envoy Gateway via Envoy Gateway Operator
Next Step
Trademark Notice

ALB Operator

Understanding ALB

Core components
Quick Start
Relationship between ALB, ALB Instance, ALB Leader
Additional resources:
Trademark Notice

Auth

Basic Concepts
Quick Start
Related Ingress Annotations
forward-auth
basic-auth
CR
ALB Special Ingress Annotation
Ingress-Nginx Auth Related Other Feature

Deploy Hig

Method 1: Use
Method 2: Use
Trademark Noti

Decision-Ma

Note: Incompatible Parts with Ingress-Ngir

Small Productic

oduc

uctic

enar

cer

Load Balancing Session Affinity Policy in ALB

Overview

Available Algorithms

Configuration Methods

Best Practices

L4/L7 Timeout

Basic Concepts

HTTP Redirect

Basic Concepts

CRD

Ingress Annotation

Port-Level Redirect

Rule-Level Redirect

Trademark Notice

CORS

Basic Concepts

CRD

Header Mod

URL Rewrite

Basic Concepts

ModSecurity

Terminology

Procedure to Operate

Related Explanations

Configuration Example

Trademark Notice

OTel

TCP/HTTP Keepalive

Basic Concepts

CRD

Trademark Notice

ALB with Ingress-NGINX Annotation Compatibility

Basic concepts

Supported ingress-nginx annotations

Trademark Notice

ALB Monitoring

Terminology

Procedure

Monitoring Metrics

Alauda Container Platform Load Balancer

TOC

- Understanding MetalLB Operator
  - Terminology
  - Principles of High Availability in MetalLB
- Algorithm for Selecting VIP Host Nodes
  - Calculation Formula
  - Application Example
- Custom Resource Definitions
  - IPAddressPool
  - L2Advertisement
  - BGPPeer
  - BGPAdvertisement
  - BFDProfile (optional)
- Constraints and Limitations
- Installing and Uninstalling the MetalLB Operator
  - Configure Component Placement
  - Adjust Log Level
- Upgrade the MetalLB Operator
- Trademark Notice

Understanding MetalLB Operator

The MetalLB Operator provides a Kubernetes-native load balancer implementation for on-premises or bare-metal environments that do not have access to cloud load balancer services. It allows Service resources of type LoadBalancer to function by assigning external IP addresses to services and advertising those addresses via Layer 2 (ARP/NDP) or Layer 3 (BGP).

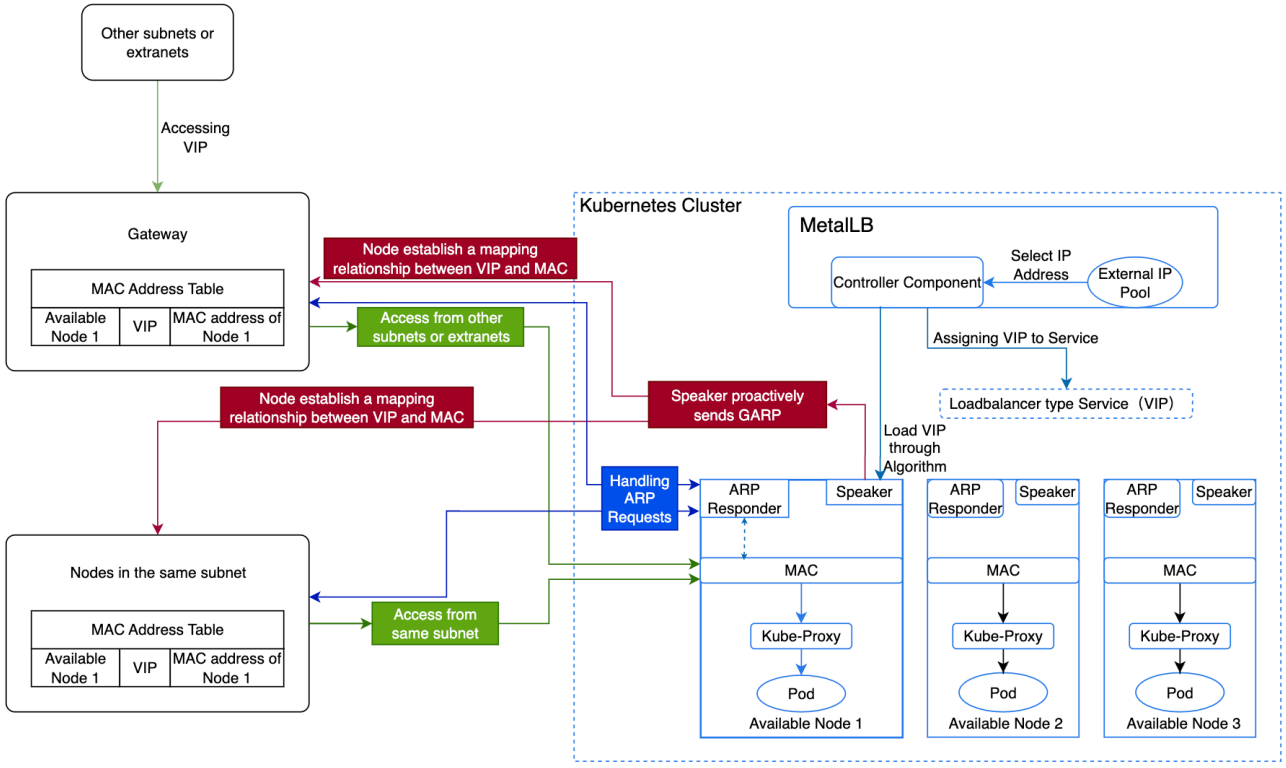
The Operator automates the lifecycle management of MetalLB components, including installation, upgrades, and configuration synchronization.

Terminology

Term	Description
VIP	A Virtual IP Address (VIP) is the IP address assigned by MetalLB for the LoadBalancer type internal routing, providing a unified access point for external traffic to access services within the cluster.
ARP	The Address Resolution Protocol (ARP) is utilized to map network layer IP addresses to data link layer MAC addresses.
GARP	Gratuitous ARP (GARP) is a special ARP request used to inform other nodes in the network about the binding of an IP address to a MAC address. Unlike normal ARP requests, GARP does not wait for responses but actively sends information across the network.
ARP Responder	A component of MetalLB responsible for responding to ARP requests by mapping the VIP to the node's MAC address. When a node needs to communicate with the VIP, it sends ARP requests to retrieve the MAC address corresponding to the VIP. Each available node has an ARP Responder that responds to these requests, mapping the VIP to the node's MAC address.
Controller	A component of MetalLB that dynamically allocates VIPs from the external address pool for LoadBalancer type internal routing. The Controller listens for creation and deletion events of internal routes in the cluster to allocate or free VIPs as required.
Speaker	A component of MetalLB that determines, based on policies or algorithms, whether nodes should host a VIP and send GARP. It ensures a certain level of

Term	Description
	balance among nodes, and when a node becomes unavailable, other nodes can take over the VIP and send GARP, thereby achieving high availability.

Principles of High Availability in MetalLB



By default, the platform uses MetalLB's ARP mode, and the specific implementation process and principles are as follows:

- The Controller component of MetalLB selects an IP address from the external address pool and allocates it to the LoadBalancer type internal routing as a VIP.
- MetalLB selects an available node as the leader to host the VIP based on the algorithm, which then forwards the traffic.
- The Speaker component on this node actively sends GARP, establishing a mapping relationship between the VIP and MAC address across all nodes.
  - Nodes within the same subnet, upon learning the mapping between the VIP and the available node's MAC address, will communicate directly with this node when accessing the VIP.
  - Nodes in different subnets will route traffic to the gateway of their subnet first, which will then forward the traffic to the node hosting the VIP.

- When this node encounters a failure, MetalLB selects another leader to host the VIP. Then send GARP to refresh Service IP mac address, thereby ensuring high availability.
- Upon reaching the node, Kube-Proxy forwards the traffic to the corresponding Pod.

Algorithm for Selecting VIP Host Nodes

The election of the "leader" (the node which is going to advertise the IP) of a given loadbalancer IP is stateless and works in the following way:

- each speaker collects the list of the potential announcers of a given IP, taking into account active speakers, external traffic policy, active endpoints, node selectors and other things.
- each speaker does the same computation: it gets a sorted list of a hash of "node+VIP" elements and announces the service if it is the first item of the list.

This removes the need of having to keep memory of which speaker is in charge of announcing a given IP.

Calculation Formula

The formula is: **Number of external address pools = ceil(n-vip / n-node)**, where ceil rounds up.

Note: If using virtual machines, the number of virtual machines = Number of external address pools * n. Here, n must be greater than 2, with a maximum of one node failure allowed.

- n-vip: Represents the number of VIPs.
- n-node: Represents the number of VIPs a single node can handle.

Application Example

If a company has 10 VIPs, and each available node can handle 5 VIPs, allowing for one node failure, how should the company plan the number of external address pools and available nodes?

Analysis:

A total of two external address pools and four available nodes are needed.

- Each available node can handle a maximum of 5 VIPs, meaning one external address pool can accommodate 5 VIPs, so two external address pools are required for 10 VIPs.
- Allowing one node failure means that each address pool must include one node hosting the VIP and one backup node, resulting in two available nodes for each of the two external address pools.

Custom Resource Definitions

The MetalLB Operator installs and manages several CRDs under the metallb.io API group. These CRDs define how MetalLB assigns and advertises IP addresses.

CRD	Kind	Purpose
`IPAddressPool`	`IPAddressPool`	Defines the pool of IP addresses available for load balancer services.
`L2Advertisement`	`L2Advertisement`	Configures how IPs from pools are advertised using ARP/NDP (Layer 2).
`BGPPeer`	`BGPPeer`	Defines a BGP peer for MetalLB to establish routing sessions (Layer 3).
`BGPAdvertisement`	`BGPAdvertisement`	Specifies which IPAddressPools are advertised over BGP and with what policies.
`BFDProfile`	`BFDProfile`	Optional; defines Bidirectional Forwarding Detection parameters for faster BGP failure detection.

IPAddressPool

Defines the range(s) of IPs MetalLB can assign to `Service.type=LoadBalancer.`

```yaml
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: production-pool
  namespace: metallb-system
spec:
  addresses: 1
    - 192.168.10.100-192.168.10.150
  autoAssign: true 2
```

1 One or more CIDRs or ranges.

2 By default, MetalLB will allocate IPs from any configured address pool with free addresses. This might end up using “expensive” addresses for services that don't require it. To prevent this behaviour you can disable automatic allocation for a pool by setting the autoAssign flag to `false`

L2Advertisement

Used in Layer 2 mode, where MetalLB announces the IP via ARP/NDP.

```yaml
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: l2-adv
  namespace: metallb-system
spec:
  ipAddressPools: 1
    - production-pool
```

1 List of pools whose IPs will be advertised.

BGPPeer

```yaml
apiVersion: metallb.io/v1beta1
kind: BGPPeer
metadata:
  name: peer1
  namespace: metallb-system
spec:
  peerAddress: 10.0.0.1 (1)
  peerASN: 64512 (2)
  myASN: 64513 (3)
  holdTime: 90s (4)
  keepaliveTime: 30s
  nodeSelectors: (5)
    - matchLabels:
        alertmanager: 'true'
```

(1) IP address of the peer router.

(2) ASN of the peer.

(3) ASN used by MetalLB.

(4) `holdTime` / `keepaliveTime`: Optional timers for BGP sessions.

(5) You can limit peers to certain nodes by using the node-selectors attribute of peers in the configuration.

BGPAdvertisement

Controls how IPs from pools are advertised over BGP.

```yaml
apiVersion: metallb.io/v1beta1
kind: BGPAdvertisement
metadata:
  name: bgp-adv
  namespace: metallb-system
spec:
  ipAddressPools: (1)
    - production-pool
  aggregationLength: 32 (2)
  communities: (3)
    - no-export
```

1. Which pools to advertise.
2. Route aggregation prefix length.
3. BGP community attributes.

BFDProfile (optional)

Used to define BFD (Bidirectional Forwarding Detection) parameters for BGP peers.

```yaml
apiVersion: metallb.io/v1beta1
kind: BFDProfile
metadata:
  name: fast-failover
spec:
  receiveInterval: 300
  transmitInterval: 300
  detectMultiplier: 3
```

Constraints and Limitations

The IP resources for the external address must meet the following conditions:

- The external address pool must be layer 2 (L2) interconnected with available nodes.
- The IPs must be usable by the platform and cannot include IPs already in use by the physical network, such as gateway IPs.
- There must be no overlap with the networks used by the cluster, including Cluster CIDR, Service CIDR, subnets, etc.
- In a dual-stack environment, ensure that both IPv4 and IPv6 addresses exist simultaneously in the same external address pool, and their counts are both greater than 0. Otherwise, dual-stack LoadBalancer type internal routes will not be able to obtain external access addresses.
- In an IPv6 environment, nodes' DNS must support IPv6; otherwise, the Alauda Container Platform Load Balancer plugin cannot be successfully deployed.

Installing and Uninstalling the MetalLB Operator

The Operator can be installed through Cluster Plugin in Alauda Container Platform web console.

1. Navigate to `Administrator` -> `Marketplace` -> `Cluster Plugin`
2. Enter the keyword `Alauda Container Platform Load Balancer`.
3. Click `⋮` then **Install**.
4. When installed, click `⋮` then **Uninstall**.

Configure Component Placement

After the Alauda Container Platform Load Balancer Cluster Plugin is installed, configure component placement directly on the `MetalLB` custom resource in the `metallb-system` namespace. The Cluster Plugin creates this resource during installation when it does not already exist; the controller and speaker scheduling settings are managed by this resource.

```yaml
apiVersion: metallb.io/v1beta1
kind: MetalLB
metadata:
  name: metallb
  namespace: metallb-system
spec:
  controllerNodeSelector:
    dedicated: ingress
  controllerTolerations:
    - key: dedicated
      operator: Equal
      value: ingress
      effect: NoSchedule
  nodeSelector:
    dedicated: ingress
  speakerTolerations:
    - key: dedicated
      operator: Equal
      value: ingress
      effect: NoSchedule
```

`controllerNodeSelector` and `controllerTolerations` apply to the MetalLB controller Deployment. `nodeSelector` and `speakerTolerations` apply to the MetalLB speaker DaemonSet. By default, these fields are empty, so MetalLB does not tolerate any additional taint.

In Layer 2 mode, the `L2Advertisement.spec.nodeSelectors` configuration determines which speakers can announce VIPs. Align it with the MetalLB component placement and the `IngressNginx.spec.controller.nodeSelector` so that the speaker can announce VIPs from nodes that run the Ingress Nginx pods.

Adjust Log Level

Set `spec.logLevel` on the `MetalLB` custom resource to adjust the log level used by both the controller and speaker:

```
spec:
  logLevel: warn
```

Supported values are `all`, `debug`, `info`, `warn`, `error`, and `none`. The default is `info`.

Upgrade the MetalLB Operator

Alauda Container Platform Load Balancer is a `Core` plugin. It is upgraded together with the cluster and does not support standalone upgrade, so you do not download or upload a package for it. For the plugin lifecycle types, see Core and Extensions.

Trademark Notice

NGINX® is a registered trademark of F5, Inc. in the U.S. and in other countries.

MetalLB™ is a trademark of The Linux Foundation.

Alauda is an independent vendor. This product is not affiliated with, endorsed by, or sponsored by The Linux Foundation or F5, Inc. All trademarks are the property of their respective owners and are used here for identification purposes only.

Alauda Container Platform Ingress Controller

TOC

- Overview
- Installation
- Configuration Via Web Console
- Configuration Via YAML
  - Ingress Sharding
    - Creating IngressNginx Which Processes Ingress In All NS
    - Creating IngressNginx Which Processes Ingress In Specific NS
  - Exposing via LoadBalancer
  - Advanced Controller Deployment Settings
  - SSL Passthrough
  - IPv4 and IPv6 Dual-Stack Support
- Additional Resources
  - Default Value Differences Compared to Official Chart
- Trademark Notice

Overview

The Ingress Nginx Operator is provided to facilitate advanced management of Ingress-Nginx deployments. This Operator streamlines the deployment, configuration, and maintenance processes of Ingress-Nginx instances. The operational architecture functions as follows: The Ingress-Nginx Operator continuously monitors Custom Resource of type `ingressnginxes.ingress-nginx.alauda.io` (IngressNginx) and automatically provisions corresponding Ingress-Nginx instances for each defined IngressNginx resource. This architecture enables direct manipulation of IngressNginx custom resources to govern Ingress-Nginx functionality and behavior.

Installation

1. Navigate to `Administrator -> Marketplace -> OperatorHub`
2. Locate the `Alauda Container Platform Ingress Controller` and click `Install`

Configuration Via Web Console

We provide basic configuration options through the user interface, which will generate the corresponding YAML for you. For more complex configurations, you can edit the YAML directly.

After completing the Installation of Ingress Nginx Operator:

1. Navigate to the `All Instances` tab
2. Click `Create`, then in the displayed dialog box, locate the IngressNginx Instance Type section and click `Create`

Field	Description	YAML Path
Name	Name of the Ingress Nginx instance	`.metadata.name`
Namespace	Namespace of the Ingress Nginx instance	`.metadata.namespace`

Field	Description	YAML Path
Replica	The replica count of the Ingress Nginx deployment	`.spec.controller.replicaCount`
Resources	The resource requests and limits of the Ingress Nginx deployment	`.spec.controller.resources`
Service Type	The service type of the Ingress Nginx	`.spec.controller.service.type`
Ingress Scope	Controls which namespaces' Ingress resources are processed	`.spec.controller.scope.namespaceSelector`, refer to ingress-sharding

Configuration Via YAML

By default, the Ingress Nginx Operator deploys Ingress Nginx instances within the same namespace as the corresponding IngressNginx custom resource.

Ingress Sharding

By default, Ingress Nginx processes Ingress resources across all namespaces for ingresses that either explicitly specify its IngressClass name or do not specify any IngressClass. You can use `.controller.scope.namespaceSelector` to restrict an instance to specific namespaces. For example, the following IngressNginx resource `demo-scope` only processes ingresses in namespaces labeled `cpaas.io/project=demo`.

Creating IngressNginx Which Processes Ingress In All NS

```yaml
apiVersion: ingress-nginx.alauda.io/v1
kind: IngressNginx
metadata:
  name: demo-all
spec:
  controller:
    nodeSelector:
      kubernetes.io/os: linux
    replicaCount: 1
```

Creating IngressNginx Which Processes Ingress In Specific NS

```yaml
apiVersion: ingress-nginx.alauda.io/v1
kind: IngressNginx
metadata:
  name: demo-scope
spec:
  controller:
    scope:
      namespaceSelector: 'cpaas.io/project=demo' (1)
    nodeSelector:
      kubernetes.io/os: linux
    replicaCount: 1
```

(1) format like `$LABLE_KEY=$LABLE_VALUE`

Exposing via LoadBalancer

By default, the Ingress Controller is configured with a ClusterIP service type. To expose the Ingress Controller externally using a LoadBalancer service, apply the following configuration:

> **NOTE**
>
> LoadBalancer Services require an external load balancer integration (cloud provider LB or MetalLB) to provision an external IP.

```yaml
apiVersion: ingress-nginx.alauda.io/v1
kind: IngressNginx
metadata:
  name: demo
spec:
  controller:
    service:
      type: LoadBalancer
```

To specify LoadBalancer VIP when using MetalLB:

```yaml
apiVersion: ingress-nginx.alauda.io/v1
kind: IngressNginx
metadata:
  name: demo
spec:
  controller:
    service:
      type: LoadBalancer
      annotations:
        metallb.universe.tf/loadBalancerIPs: '192.168.2.2' # Desired VIP
        metallb.universe.tf/address-pool: 'pool-name' # MetalLB address p
ool
```

Advanced Controller Deployment Settings

Configure network mode, replicas, resource limits, node selection, and taint toleration:

```yaml
apiVersion: ingress-nginx.alauda.io/v1
kind: IngressNginx
metadata:
  name: demo
spec:
  controller:
    hostNetwork: false
    replicaCount: 1
    nodeSelector:
      dedicated: ingress
    tolerations:
      - key: dedicated
        operator: Equal
        value: ingress
        effect: NoSchedule
    resources:
      limits:
        cpu: 200m
        memory: 256Mi
      requests:
        cpu: 200m
        memory: 256Mi
```

The `nodeSelector` and `tolerations` settings control where the Ingress Nginx controller pods are scheduled. By default, the Ingress Nginx controller tolerates no taints.

SSL Passthrough

Enable SSL passthrough functionality:

CAUTION

With SSL passthrough, TLS terminates on the backend, so L7 features (e.g., request/response header manipulation, WAF, HTTP-to-HTTPS redirect, some auth flows) will not apply to that traffic at the controller.

```yaml
apiVersion: ingress-nginx.alauda.io/v1
kind: IngressNginx
metadata:
  name: demo
spec:
  controller:
    extraArgs:
      enable-ssl-passthrough: ''
```

IPv4 and IPv6 Dual-Stack Support

```yaml
apiVersion: ingress-nginx.alauda.io/v1
kind: IngressNginx
metadata:
  name: demo
spec:
  controller:
    service:
      ipFamilyPolicy: PreferDualStack
```

Additional Resources

The `.spec` field of the `IngressNginx` resource directly corresponds to the Helm chart values for Ingress Nginx, including `.spec.controller.nodeSelector` and `.spec.controller.tolerations`. For additional configuration options, please refer to the official Ingress NGINX documentation ↗.

Default Value Differences Compared to Official Chart

1. By default, each IngressNginx instance creates an IngressClass with the name `$ns-$name` and controllerValue `ingress-nginx.cpaas.io/$ns-$name`. These values can be customized using the `.spec.controller.ingressClassResource.name` and `.spec.controller.ingressClassResource.controllerValue` parameters.
2. By default, the `.spec.controller.service.type` is set to `ClusterIP`.

3. By default, `.spec.controller.watchIngressWithoutClass` is set to `true`, which means the controller will process Ingress resources that do not specify an IngressClass.

Trademark Notice

NGINX® is a registered trademark of F5, Inc. in the U.S. and in other countries.

MetalLB™ is a trademark of The Linux Foundation.

Alauda is an independent vendor. This product is not affiliated with, endorsed by, or sponsored by The Linux Foundation or F5, Inc. All trademarks are the property of their respective owners and are used here for identification purposes only.

Alauda Container Platform Gateway API Controller

TOC

- Overview
  - Architecture
  - Related custom resources
    - Operator-related custom resources
    - `Gateway API` related custom resources
    - Envoy Gateway deploy-related custom resources
- Install Envoy Gateway via Envoy Gateway Operator
  - Prerequisites
  - Installation
    - Step 1: Install Envoy Gateway Operator
    - Step 2: Create an EnvoyGatewayCtl with the Recommended Default Values
  - Configuration Via YAML
  - Advanced Config Via EnvoyGatewayCtl
    - Example: Enable Backend Routing
- Next Step
- Trademark Notice

Overview

Use `envoy-gateway-operator` to deploy and manage Envoy Gateway through platform-managed resources. The main custom resources and runtime instances are `EnvoyGatewayCtl`, `GatewayClass`, `Gateway`, `EnvoyProxy`, `envoy-gateway instance`, and `envoy-proxy instance`.

The `envoy-gateway-operator` packages the upstream `envoy-gateway` Helm chart as an Operator, so you can deploy and manage an `envoy-gateway instance` declaratively through the `EnvoyGatewayCtl` custom resource instead of managing the chart or rendered resources directly.

In ACP 4.3, `envoy-gateway-operator` packages the upstream `envoy-gateway` Helm chart version `1.7.0`.

For the Gateway API CRD versions installed by each ACP release, see Gateway API CRD versions.

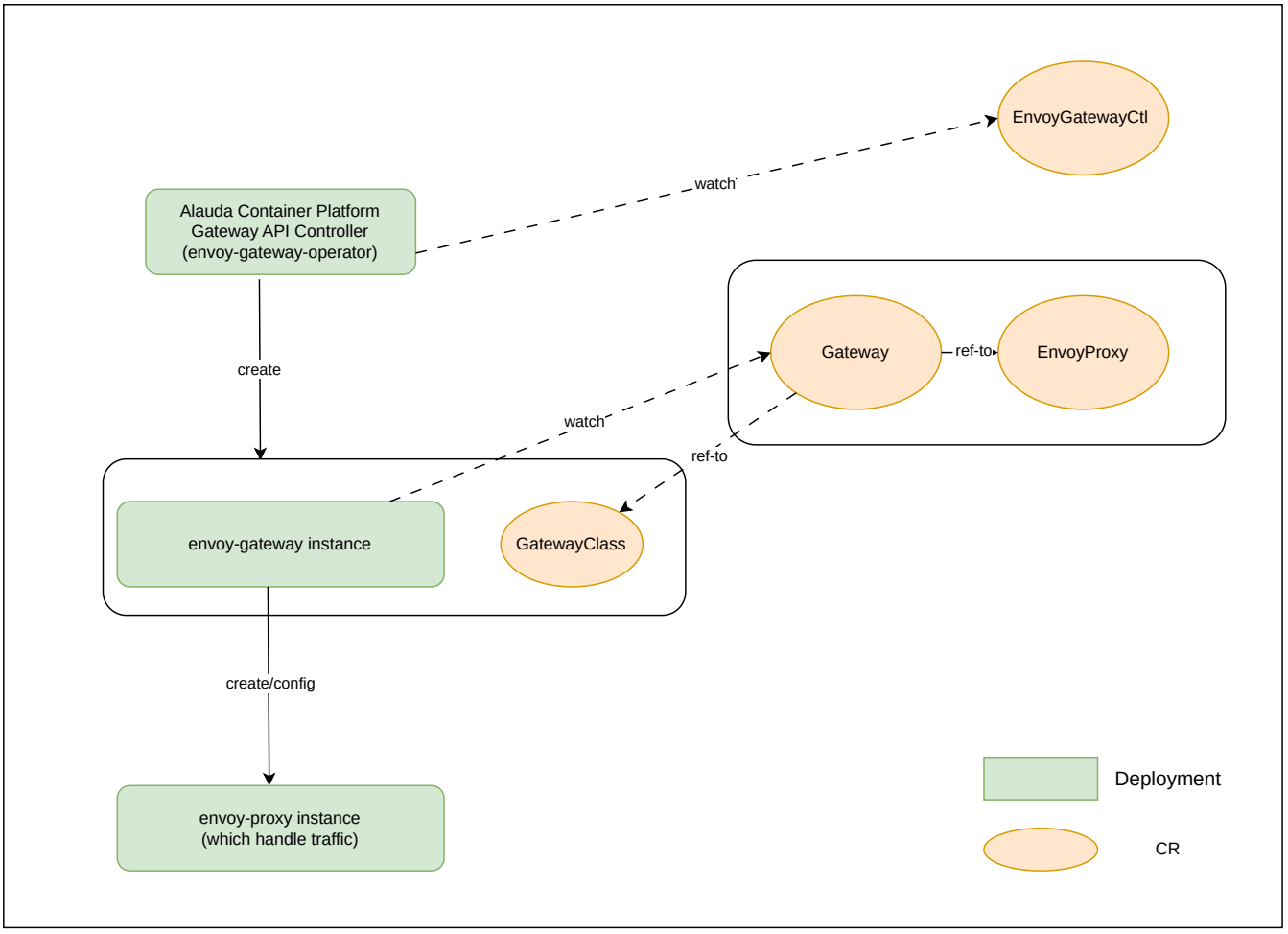
Because the Operator is maintained by the platform, it is also upgraded automatically together with the platform, which helps reduce the operational overhead of manually maintaining the Envoy Gateway installation.

Create an `EnvoyGatewayCtl` as the entry point for deploying and managing Envoy Gateway through the platform.

After you understand these concepts and create an `EnvoyGatewayCtl`, continue with `Gateway`, `Route`, and `Policy` configuration in the order listed in Next Step.

Architecture

The following workflow shows how `envoy-gateway-operator`, `EnvoyGatewayCtl`, `GatewayClass`, `Gateway`, and `EnvoyProxy` work together to deploy and manage Envoy Gateway and its data plane.



1. After you install `envoy-gateway-operator` and create an `EnvoyGatewayCtl`, the operator deploys an `envoy-gateway instance` and creates a corresponding `GatewayClass`. The `GatewayClass` name follows the `<namespace>-<name>` pattern. The recommended default example uses `cpaas-default` in the `envoy-gateway-operator` namespace, which is also the default configuration prefilled on the `Create EnvoyGatewayCtl` page. In this case, the generated `GatewayClass` is `envoy-gateway-operator-cpaas-default`.
   Each `EnvoyGatewayCtl` must have a unique combination of namespace and name, so the corresponding `GatewayClass` names are also unique. When creating a `Gateway`, you select the appropriate `GatewayClass` to determine which `envoy-gateway instance` manages it.
2. When you create a `Gateway` that references this `GatewayClass`, the `envoy-gateway instance` takes ownership of that `Gateway`. If the `Gateway` references an `EnvoyProxy` through `.spec.infrastructure.parametersRef`, that `EnvoyProxy` is used to control how the underlying `envoy-proxy instance` is deployed.
3. The `envoy-gateway instance` then provisions and manages the corresponding `envoy-proxy instance`, which serves as the data plane and handles incoming traffic for that `Gateway`.

4. You can create `HTTPRoute`, `TCPRoute`, `UDPRoute`, `GRPCRoute`, or `TLSRoute` resources and attach them to the `Gateway` to define how traffic is matched and forwarded to backend Services. You can also attach traffic policies such as `ClientTrafficPolicy` or `BackendTrafficPolicy` for more advanced behavior.

Related custom resources

The following custom resources are involved in deploying Envoy Gateway and configuring traffic behavior.

Operator-related custom resources

Custom resources	Description
EnvoyGatewayCtl	Controls the deployment of a specific `envoy-gateway instance`.

We recommend deploying a single `envoy-gateway instance` via `EnvoyGatewayCtl` per cluster for typical use cases.

For advanced scenarios, `EnvoyGatewayCtl` also allows you to deploy multiple `envoy-gateway instance` resources within the same cluster for stronger control and isolation, such as different deployment mode ↗.

`Gateway API` related custom resources

These resources describe the routing semantics of the gateway, including how traffic enters the gateway, how requests are matched and forwarded to backends, and what policies are applied during traffic processing.

Custom resources	Description
GatewayClass ↗	Tells an `envoy-gateway instance` which `Gateway` resources it should manage.
Gateway ↗	Defines three essential components: 1. **GatewayClass reference** - References the GatewayClass that controls this Gateway via `.spec.gatewayClassName`

Custom resources	Description
	2. **Listeners configuration** - Specifies which ports, hostnames, and TLS certificates to use for traffic handling via `.spec.listeners` 3. **Infrastructure parameters** - Configures deployment specifics like replicas, resources by referencing an `EnvoyProxy` via `.spec.infrastructure.parametersRef`
Route resources ↗	Includes `HTTPRoute`, `TCPRoute`, `UDPRoute`, `GRPCRoute`, and `TLSRoute`. These resources define four essential components: 1. **Multiple matching rule sets** - Various conditions to match incoming traffic 2. **Backend destinations** - Target services for each matching rule set 3. **Traffic policies** - Per-rule configurations including retries, timeouts, redirects, etc. 4. **Gateway listeners attachment** - Defined via `.spec.parentRefs` to connect routes to specific `Gateway` listeners
Policies	Policy resources extend `Gateway` and `Route` resources with advanced capabilities. Envoy Gateway currently provides `SecurityPolicy`, `BackendTLSPolicy`, `ClientTrafficPolicy`, and `BackendTrafficPolicy`.

For more details about these standard Gateway API resources, please refer to the Gateway API Concepts and Configure GatewayAPI Policy.

Envoy Gateway deploy-related custom resources

These resources control how Envoy Gateway and its data plane are deployed. Use the following custom resource for deployment-related configuration.

Custom resources	Description
EnvoyProxy ↗	Defines the configuration for `envoy-proxy instances`, such as replicas, resources, and scheduling constraints

To limit the total CPU and memory requests declared by `EnvoyProxy` resources in a namespace, see Configure EnvoyProxy Resource Quotas.

In the recommended deployment pattern, each `Gateway` references its own dedicated `EnvoyProxy` through `.spec.infrastructure.parametersRef`. When you create a `Gateway` from the Web Console by using an `EnvoyGatewayCtl`-created `GatewayClass`, the console automatically creates a companion `EnvoyProxy` resource with the same name and namespace. This one-to-one mapping lets you control the Gateway's deployment configuration, such as replicas, resources, and scheduling, by updating the corresponding `EnvoyProxy` resource.

Install Envoy Gateway via Envoy Gateway Operator

Prerequisites

Before proceeding, review the concepts and architecture above.

Installation

Step 1: Install Envoy Gateway Operator

1. Navigate to `Administrator -> Marketplace -> OperatorHub`
2. Locate the `Alauda Container Platform Gateway API Controller` under `Networking` Category and click `Install`

Step 2: Create an EnvoyGatewayCtl with the Recommended Default Values

The following default values match the recommended example used in this workflow.

1. Navigate to `Administrator -> Marketplace -> OperatorHub`
2. Locate the `Alauda Container Platform Gateway API Controller` under the `Networking` category, then click it to open the details page.
3. Navigate to the `All Instances` tab

4. Click `Create`, then in the displayed dialog box, locate the `EnvoyGatewayCtl` Instance Type section and click `Create`
5. The `Create EnvoyGatewayCtl` page displays the following configuration options:

Field	Default Value	Description	YAML Path
Name	cpaas-default	Name of the Envoy Gateway instance	`.metadata.name`
Namespace	envoy-gateway-operator	Namespace of the Envoy Gateway instance. Each namespace can have only one `EnvoyGatewayCtl`.	`.metadata.namespace`
Replica	1	Replica count of the Envoy Gateway deployment	`.spec.deployment.replic`
Deploy Ns Mode	ControllerNamespace	Determines whether the gateway deployment is installed in the gateway namespace or the controller namespace. For details, see deployment mode ↗	`.spec.config.envoyGatew`
Resources	1C1G	Resource requests and limits of the Envoy Gateway deployment	`.spec.deployment.resour`

Recommendation: Use the default values for typical deployments. Click **Create** to proceed.

Configuration Via YAML

As an alternative to the Web Console, you can create an `EnvoyGatewayCtl` by applying YAML.

Default Config:

```yaml
apiVersion: envoy-gateway.alauda.io/v1
kind: EnvoyGatewayCtl
metadata:
  name: cpaas-default
  namespace: envoy-gateway-operator
spec:
  config:
    envoyGateway:
      provider:
        kubernetes:
          deploy:
            type: ControllerNamespace
        type: Kubernetes
  deployment:
    replicas: 1
    resources:
      limits:
        cpu: '1'
        memory: 1024Mi
      requests:
        cpu: 100m
        memory: 256Mi
```

Advanced Config Via EnvoyGatewayCtl

`EnvoyGatewayCtl` is the custom resource used to configure an `envoy-gateway instance`. Its `spec` maps directly to the values of the upstream `envoy-gateway` Helm chart. When you need to modify `EnvoyGatewayCtl`, refer to the upstream chart values documentation for the available configuration options.

For detailed information about available configuration options, please refer to envoy-gateway-chart-values↗.

> **DANGER**
>
> You should only configure the `envoy-gateway instance` through the `EnvoyGatewayCtl` resource. Do not directly modify rendered resources such as ConfigMaps, Deployments, as these changes may be lost during upgrades or reconciliation.

Example: Enable Backend Routing

The following example shows how to enable backend routing in the `EnvoyGatewayCtl` spec:

```yaml
apiVersion: envoy-gateway.alauda.io/v1
kind: EnvoyGatewayCtl
metadata:
  name: cpaas-default
  namespace: envoy-gateway-operator
spec:
  config:
    envoyGateway:
      extensionApis:
        enableBackend: true # Enable backend routing in EnvoyGatewayCtl n
ot the envoygateway configmap
      provider:
        kubernetes:
          deploy:
            type: ControllerNamespace
        type: Kubernetes
  deployment:
    replicas: 1
    resources:
      limits:
        cpu: '1'
        memory: 1024Mi
      requests:
        cpu: 100m
        memory: 256Mi
```

Next Step

After the Envoy Gateway operator and `EnvoyGatewayCtl` are ready, continue with the following tasks in order:

- Configure GatewayAPI Gateway
- Configure GatewayAPI Route
- Configure GatewayAPI Policy
- Tasks for Envoy Gateway

Trademark Notice

Envoy® is a registered trademark of The Linux Foundation.

Alauda is an independent vendor. This product is not affiliated with, endorsed by, or sponsored by The Linux Foundation. All trademarks are the property of their respective owners and are used here for identification purposes only.

ALB Operator

Understanding ALB

Core components

Quick Start

Relationship between ALB, ALB Instance, ALB Leader

Additional resources:

Trademark Notice

Auth

Basic Concepts

Quick Start

Related Ingress Annotations

forward-auth

basic-auth

CR

ALB Special Ingress Annotation

Ingress-Nginx Auth Related Other Feature

Note: Incompatible Parts with Ingress-Ngir

Deploy High

Method 1: Use

Method 2: Use

Trademark Noti

Decision-Ma

Small Productio

oduc

uctio

enar

cer

Load Balancing Session Affinity Policy in ALB

Overview

Available Algorithms

Configuration Methods

Best Practices

L4/L7 Timeout

Basic Concepts

HTTP Redirect

Basic Concepts

CRD

Ingress Annotation

CORS

Basic Concepts

CRD

Port-Level Redirect

Rule-Level Redirect

Trademark Notice

URL Rewrite

Basic Concepts

Header Mod

ModSecurity

Terminology

Procedure to Operate

Related Explanations

Configuration Example

Trademark Notice

OTel

TCP/HTTP Keepalive

Basic Concepts

CRD

Trademark Notice

ALB with Ingress-NGINX Annotation Compatibility

Basic concepts

Supported ingress-nginx annotations

Trademark Notice

ALB Monitoring

Terminology

Procedure

Monitoring Metrics

Understanding ALB

WARNING

ALB has been deprecated. Please use the ingress-nginx-operator or envoy-gateway instead.

ALB (Another Load Balancer) is a Kubernetes Gateway powered by OpenResty with years of production experience from Alauda.

TOC

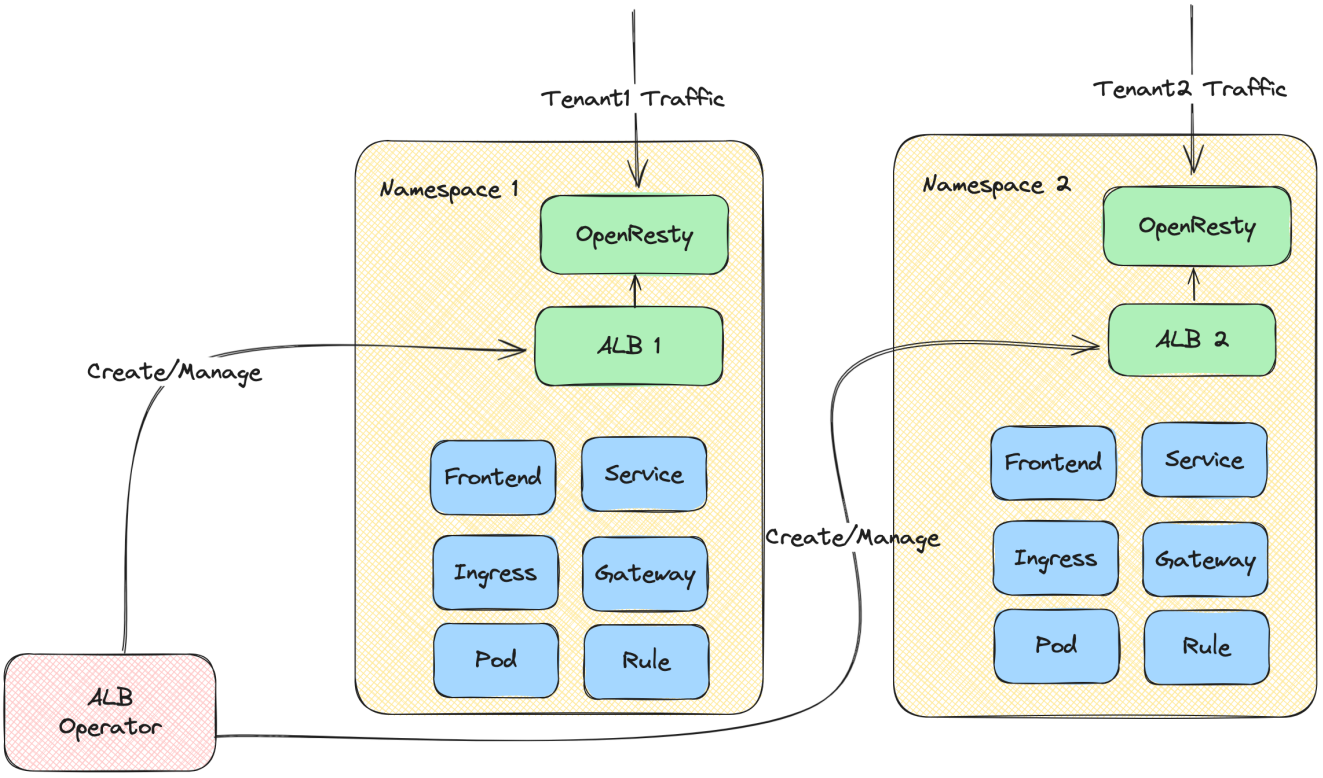
- Core components
- Quick Start
  - Deploy the ALB Operator
  - Deploy an ALB Instance
  - Run a demo application
- Relationship between ALB, ALB Instance, Frontend/FT, Rule, Ingress, and Project
  - Ingress
  - Ingress Controller
  - ALB
  - ALB Instance
  - ALB-Operator
  - Frontend (abbreviation: FT)
  - RULE
- ALB Leader

Project

Additional resources:

Trademark Notice

Core components



- **ALB Operator**: An operator that manages the lifecycle of ALB instances. It watches ALB CRs and creates/updates instances for different tenants.
- **ALB Instance**: An ALB instance includes OpenResty acting as the data plane and a Go controller as the control plane. The Go controller monitors various CRs (Ingress, Gateway, Rule, etc.) and converts them into ALB-specific DSL rules. OpenResty then uses these DSL rules to match and process incoming requests.

Quick Start

Deploy the ALB Operator

1. Create a cluster.

2. ```
   helm repo add alb https://alauda.github.io/alb/;helm repo update;helm search repo|grep alb
   ```

3. ```
   helm install alb-operator alb/alauda-alb2
   ```

Deploy an ALB Instance

```
cat <<EOF | kubectl apply -f -
apiVersion: crd.alauda.io/v2beta1
kind: ALB2
metadata:
    name: alb-demo
    namespace: kube-system
spec:
    address: "172.20.0.5"  # the ip address of node where alb been deployed
    type: "nginx"
    config:
        networkMode: host
        loadbalancerName: alb-demo
        projects:
        - ALL_ALL
        replicas: 1
EOF
```

Run a demo application

```
cat <<EOF | kubectl apply -f -
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-world
  labels:
    k8s-app: hello-world
spec:
  replicas: 1
  selector:
    matchLabels:
      k8s-app: hello-world
  template:
    metadata:
      labels:
        k8s-app: hello-world
    spec:
      terminationGracePeriodSeconds: 60
      containers:
      - name: hello-world
        image: crccheck/hello-world:latest
        imagePullPolicy: IfNotPresent
---
apiVersion: v1
kind: Service
metadata:
  name: hello-world
  labels:
    k8s-app: hello-world
spec:
  ports:
  - name: http
    port: 80
    targetPort: 8000
  selector:
    k8s-app: hello-world
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: hello-world
spec:
  rules:
```

```
  - http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: hello-world
            port:
              number: 80
EOF
```

Now you can access the app via `curl http://${ip}`

Relationship between ALB, ALB Instance, Frontend/FT, Rule, Ingress, and Project

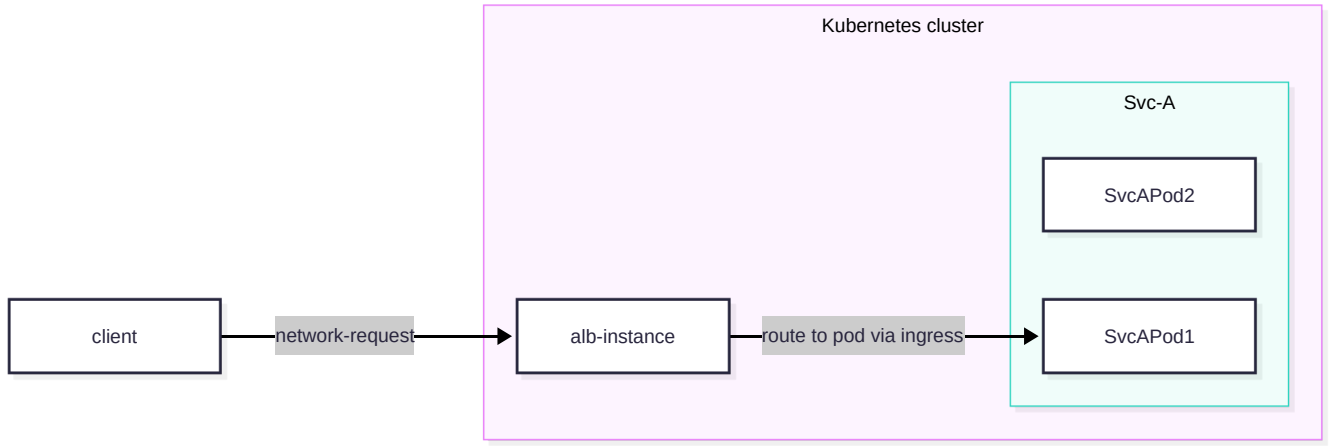
LoadBalancer is a key component in modern cloud-native architectures, serving as an intelligent traffic router and load balancer.

To understand how ALB works in a Kubernetes cluster, we need to understand several core concepts and their relationships:

- ALB itself
- Frontend (FT)
- Rules
- Ingress resources
- Projects

These components work together to enable flexible and powerful traffic management capabilities.

The following explains how these concepts work together in the request path. Detailed introductions for each concept are covered in separate articles.



In a request-calling chain:

1. A client sends an HTTP/HTTPS/other protocol request, and finally the request will **arrive on a pod of ALB**, and the pod (an ALB instance) will start to handle this request.
2. This ALB instance finds a rule which could match this request.
3. If needed, modify/redirect/rewrite the request based on the rule.
4. Find and select one pod IP from the services which the rule configured. And forward the request to the pod.

Ingress

Ingress is a resource in Kubernetes, used to describe what request should be sent to which service.

Ingress Controller

A program that understands Ingress resource and will proxy request to service.

ALB

ALB is an Ingress controller.

In Kubernetes cluster, we use the `alb2` resource to operate an ALB. You could use `kubectl get alb2 -A` to view all the ALBs in the cluster.

ALBs are created by users manually. Each ALB has its own IngressClass. When you create an Ingress, you can use `.spec.ingressClassName` field to indicate which Ingress controller

should handle this Ingress.

ALB Instance

ALB also is a Deployment (bunch of pods) running in the cluster. Each pod is called an ALB instance.

Each ALB instance handles requests independently, but all instances share Frontend (FT), Rule, and other configurations belonging to the same ALB.

ALB-Operator

ALB-Operator, a default component deployed in the cluster, is an operator for ALB. It will create/update/delete Deployment and other related resources for each ALB according to the ALB resource.

Frontend (abbreviation: FT)

FT is a resource defined by ALB itself. It is used to represent the ALB instance listening ports.

FT could be created by ALB-Leader or user manually.

Cases of FT created by ALB-Leader:

1. If Ingress has certificate, we will create FT 443 (HTTPS).
2. If Ingress has no certificate, we will create FT 80 (HTTP).

RULE

RULE is a resource defined by ALB itself. It takes the same role as the Ingress, but it is more specific. A RULE is uniquely associated with a FT.

RULE could be created by ALB-Leader or user manually.

Cases of RULE created by ALB-Leader:

1. Sync Ingress to RULE.

ALB Leader

In multiple ALB instances, one will be elected as leader. The leader is responsible for:

1. Translating the Ingress into Rules. We will create Rule for each path in the Ingress.
2. Creating FT needed by Ingress. For example, if Ingress has certificate we will create FT 443 (HTTPS), if Ingress has no certificate we will create FT 80 (HTTP).

Project

From the perspective of ALB, Project is a set of namespaces.

You could configure one or more Projects in an ALB. When ALB Leader translates the Ingress into Rules, it will ignore Ingress in namespaces which do not belong to the Project.

Additional resources:

- Configure a Load Balancer

Trademark Notice

Envoy® is a registered trademark of The Linux Foundation.

NGINX® is a registered trademark of F5, Inc. in the U.S. and in other countries.

Alauda is an independent vendor. This product is not affiliated with, endorsed by, or sponsored by The Linux Foundation or F5, Inc. All trademarks are the property of their respective owners and are used here for identification purposes only.

Auth

TOC

- Basic Concepts
  - What is Auth
  - Supported Auth Methods
  - Auth Configuration Methods
  - Auth Result Handling
- Quick Start
  - Deploy ALB
  - Configure Secret and Ingress
  - Verify
- Related Ingress Annotations
- forward-auth
  - Construct Related Annotations
    - auth-url
    - auth-method
    - auth-proxy-set-headers
  - Construct app-request related annotations
    - auth-response-headers
  - cookie handling
  - Redirect sign related configuration
    - auth-signin

auth-signin-redirect-param
auth-request-redirect
basic-auth
auth-realm
auth-type
auth-secret
auth-secret-type
CR
ALB Special Ingress Annotation
Auth-Enable
Ingress-Nginx Auth Related Other Features
Global-Auth
No-Auth-Locations
Note: Incompatible Parts with Ingress-Nginx
Troubleshooting
Trademark Notice

Basic Concepts

What is Auth

Auth is a mechanism that performs authentication before a request reaches the actual service. It allows you to handle authentication at the ALB level uniformly, without implementing authentication logic in each backend service.

Supported Auth Methods

ALB supports two main authentication methods:

1. **Forward Auth (External Authentication)**

- Send a request to an external authentication service to verify the user's identity

- Applicable scenarios: Need complex authentication logic, such as OAuth, SSO, etc.
- Workflow:
  1. User request arrives at ALB
  2. ALB forwards the authentication information to the authentication service
  3. The authentication service returns the verification result
  4. Based on the authentication result, decide whether to allow access to the backend service

2. **Basic Auth (Basic Authentication)**

- A simple authentication mechanism based on username and password
- Applicable scenarios: Simple access control, development environment protection
- Workflow:
  1. User request arrives at ALB
  2. ALB checks the username and password in the request
  3. Compare with the configured authentication information
  4. If the verification passes, forward to the backend service

Auth Configuration Methods

1. **Global Auth**

- Configure at the ALB level, applicable to all services
- Configure at the ALB or FT CR

2. **Path-level Auth**

- Configure at the specific Ingress path
- Configure at the specific Rule
- Can override the global auth configuration

3. **Disable Auth**

- Disable auth for a specific path

- Configure at the Ingress with annotation: `alb.ingress.cpaas.io/auth-enable: "false"`
- Configure at the Rule with CR

Auth Result Handling

- Auth success: Request will be forwarded to the backend service
- Auth failed: Return 401 unauthorized error
- Can configure the redirect behavior after auth failed (applicable to Forward Auth)

Quick Start

Configure Basic Auth with ALB

Deploy ALB

```
cat <<EOF | kubectl apply -f -
apiVersion: crd.alauda.io/v2
kind: ALB2
metadata:
  name: auth
  namespace: cpaas-system
spec:
  config:
    networkMode: container
    projects:
    - ALL_ALL
    replicas: 1
    vip:
      enableLbSvc: false
  type: nginx
EOF
export ALB_IP=$(kubectl get pods -n cpaas-system -l service_name=alb2-aut
h -o jsonpath='{.items[*].status.podIP}');echo $ALB_IP
```

Configure Secret and Ingress

```
# echo "Zm9vOiRhcHIxJHFJQ05aNjFRJDJpb29pSlZVQU1tcHJxMjU4L0NoUDE=" | base6
4 -d #  foo:$apr1$qICNZ61Q$2iooiJVUAMmprq258/ChP1
# openssl passwd -apr1 -salt qICNZ61Q bar # $apr1$qICNZ61Q$2iooiJVUAMmprq
258/ChP1

kubectl apply -f - <<'END'
apiVersion: v1
kind: Secret
metadata:
  name: auth-file
type: Opaque
data:
  auth: Zm9vOiRhcHIxJHFJQ05aNjFRJDJpb29pSlZVQU1tcHJxMjU4L0NoUDE=
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: auth-file
  annotations:
    "nginx.ingress.kubernetes.io/auth-type": "basic"
    "nginx.ingress.kubernetes.io/auth-secret": "default/auth-file"
    "nginx.ingress.kubernetes.io/auth-secret-type": "auth-file"
spec:
  rules:
  - http:
      paths:
      - path: /app-file
        pathType: Prefix
        backend:
          service:
            name: app-server
            port:
              number: 80
END
```

Verify

```
# echo "Zm9vOiJhYXIi" | base64 -d # foo:bar
curl -v -X GET -H "Authorization: Basic Zm9vOmJhcg==" http://$ALB_IP:80/a
pp-file # should return 200
# wrong password
curl -v -X GET -H "Authorization: Basic XXXXOmJhcg==" http://$ALB_IP:80/a
pp-file # should return 401
```

Related Ingress Annotations

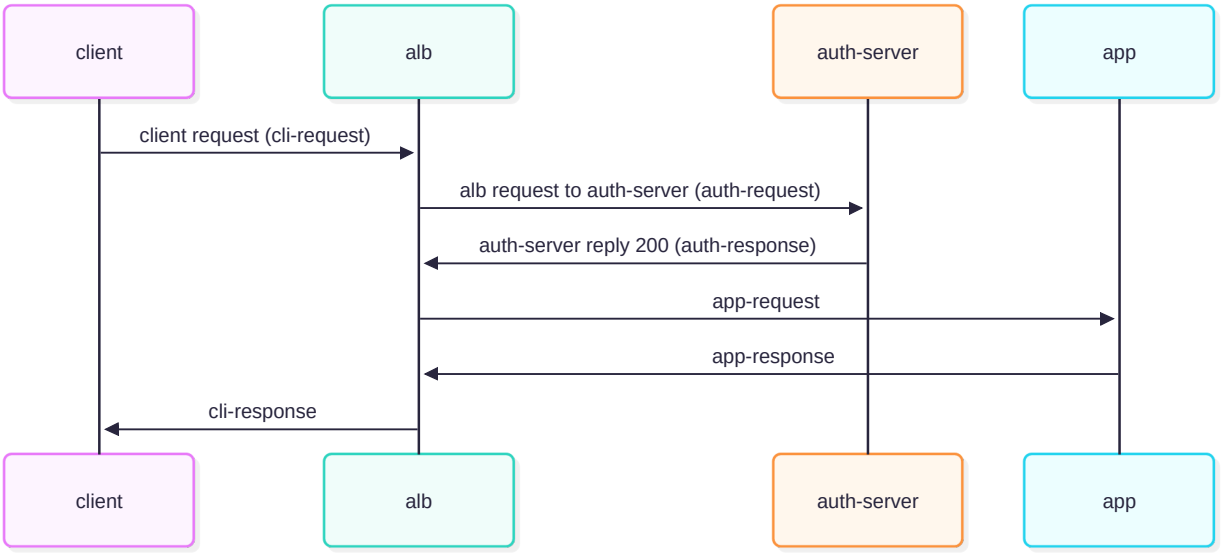
Ingress-nginx defines a series of annotations to configure the details of the authentication process. Below is a list of annotations that ALB supports, where "v" indicates support and "x" indicates no support.

	support	type	note
forward-auth			forward auth by sending http request
nginx.ingress.kubernetes.io/auth-url	v	string	
nginx.ingress.kubernetes.io/auth-method	v	string	
nginx.ingress.kubernetes.io/auth-signin	v	string	
nginx.ingress.kubernetes.io/auth-signin-redirect-param	v	string	
nginx.ingress.kubernetes.io/auth-response-headers	v	string	
nginx.ingress.kubernetes.io/auth-proxy-set-headers	v	string	

	support	type	note
nginx.ingress.kubernetes.io/auth-request-redirect	v	string	
nginx.ingress.kubernetes.io/auth-always-set-cookie	v	boolean	
nginx.ingress.kubernetes.io/auth-snippet	x	string	
basic-auth			auth by username and password secret
nginx.ingress.kubernetes.io/auth-realm	v	string	
nginx.ingress.kubernetes.io/auth-secret	v	string	
nginx.ingress.kubernetes.io/auth-secret-type	v	string	
nginx.ingress.kubernetes.io/auth-type	-	"basic" or "digest"	basic: supports apr1 **digest: not supported**
auth-cache			
nginx.ingress.kubernetes.io/auth-cache-key	x	string	
nginx.ingress.kubernetes.io/auth-cache-duration	x	string	
auth-keepalive			keepalive when sending request. specify keepalive behavior through a series of annotations

	support	type	note
nginx.ingress.kubernetes.io/auth-keepalive	x	number	
nginx.ingress.kubernetes.io/auth-keepalive-share-vars	x	"true" or "false"	
nginx.ingress.kubernetes.io/auth-keepalive-requests	x	number	
nginx.ingress.kubernetes.io/auth-keepalive-timeout	x	number	
auth-tls ↗			when request is https, extra verify the certificate.
nginx.ingress.kubernetes.io/auth-tls-secret	x	string	
nginx.ingress.kubernetes.io/auth-tls-verify-depth	x	number	
nginx.ingress.kubernetes.io/auth-tls-verify-client	x	string	
nginx.ingress.kubernetes.io/auth-tls-error-page	x	string	
nginx.ingress.kubernetes.io/auth-tls-pass-certificate-to-upstream	x	"true" or "false"	
nginx.ingress.kubernetes.io/auth-tls-match-cn	x	string	

forward-auth



Related annotations:

- nginx.ingress.kubernetes.io/auth-url
- nginx.ingress.kubernetes.io/auth-method
- nginx.ingress.kubernetes.io/auth-signin
- nginx.ingress.kubernetes.io/auth-signin-redirect-param
- nginx.ingress.kubernetes.io/auth-response-headers
- nginx.ingress.kubernetes.io/auth-proxy-set-headers
- nginx.ingress.kubernetes.io/auth-request-redirect
- nginx.ingress.kubernetes.io/auth-always-set-cookie

These annotations describe the modifications made to auth-request, app-request, and cli-response in the above diagram.

Construct Related Annotations

auth-url

Auth-request's URL, value can be a variable.

auth-method

Auth-request's method.

auth-proxy-set-headers

The value is a ConfigMap reference in the format `ns/name`. By default, all headers from the cli-request will be sent to the auth-server. Additional headers can be configured through proxy_set_header. The following headers are sent by default:

```
X-Original-URI          $request_uri;
X-Scheme                $pass_access_scheme;
X-Original-URL          $scheme://$http_host$request_uri;
X-Original-Method       $request_method;
X-Sent-From             "alb";
X-Real-IP               $remote_addr;
X-Forwarded-For         $proxy_add_x_forwarded_for;
X-Auth-Request-Redirect $request_uri;
```

Construct app-request related annotations

auth-response-headers

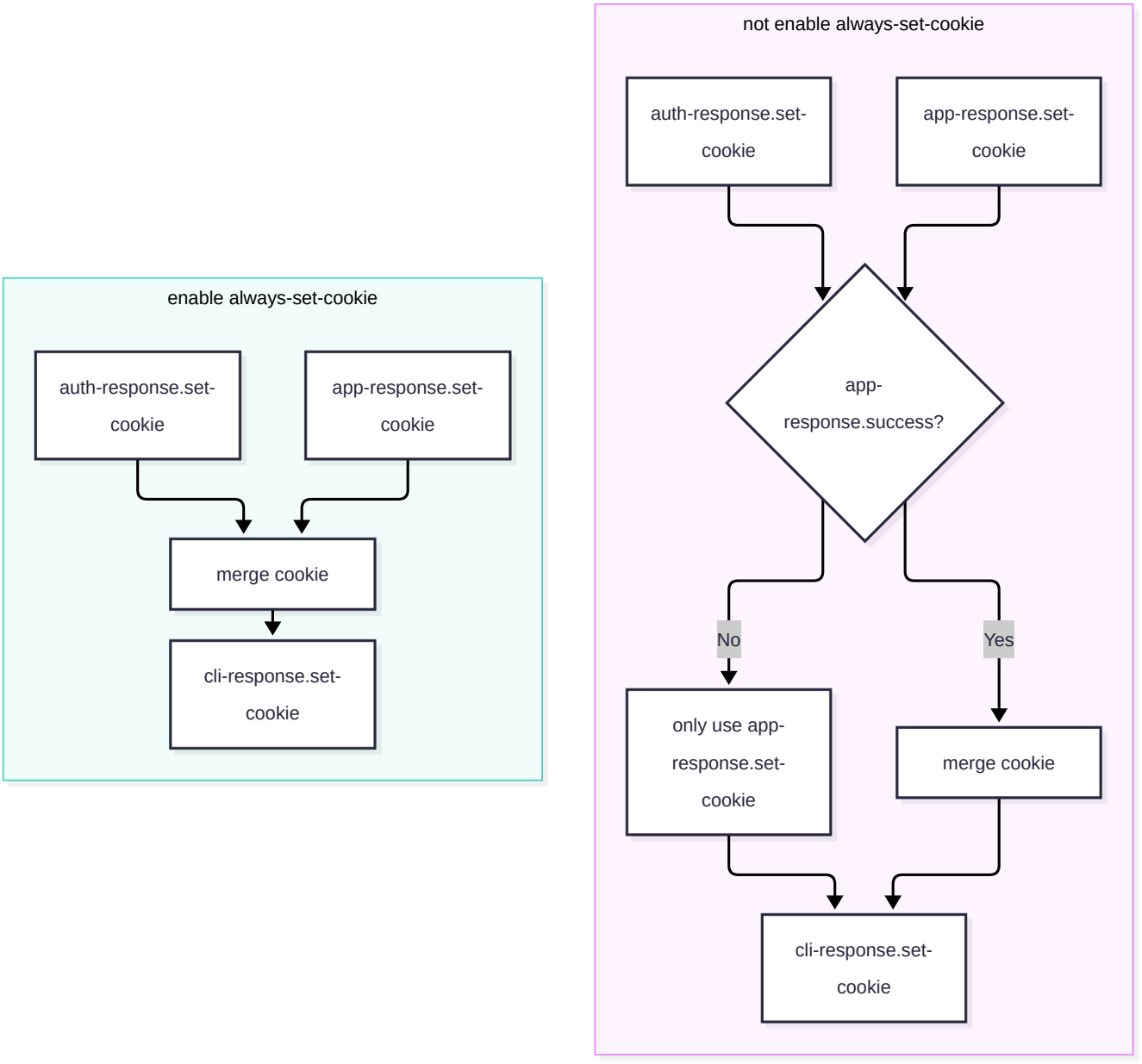
Value is a comma-separated string, allowing us to bring specific headers from auth-response to app-request. example:

```
nginx.ingress.kubernetes.io/auth-response-headers: Remote-User,Remote-Nam
e
```

When ALB initiates an app-request, it will include the Remote-User and Remote-Name from the auth-response headers.

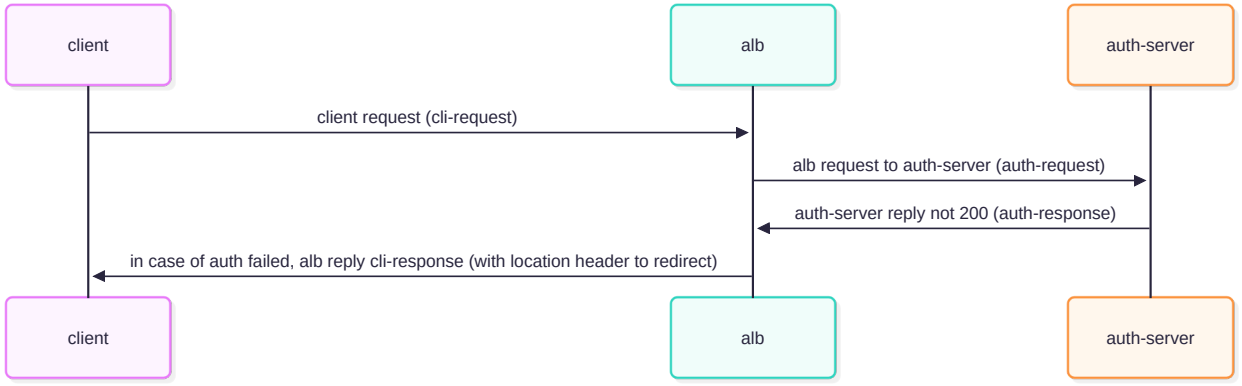
cookie handling

auth-response and app-response can both set cookies. By default, only when app-response.success, the auth-response.set-cookie will be merged into cli-response.set-cookie.



Redirect sign related configuration

When the auth-server returns 401, we can set the redirect header in the cli-response to instruct the browser to redirect to the url specified by auth-signin for verification.



auth-signin

Value is a url, specify the location header in cli-response.

auth-signin-redirect-param

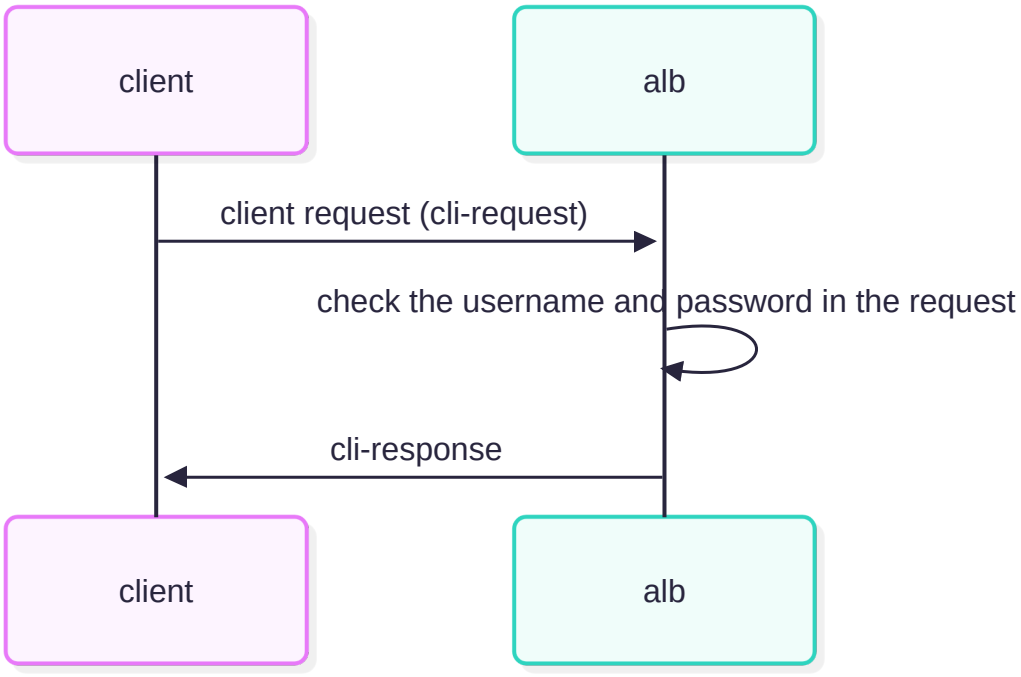
The name of the query parameter in the signin-url, default is rd. if the signin-url does not contain the `auth-signin-redirect-param` specified parameter name, alb will automatically add the parameter. The parameter value will be set to `$pass_access_scheme://$http_host$escaped_request_uri`, used to record the original request URL.

auth-request-redirect

Set the `x-auth-request-redirect` header in auth-request.

basic-auth

basic-auth is the authentication process described in RFC 7617↗. The interaction process is as follows:



auth-realm

description of the protected area↗ Which is the realm value in the `WWW-Authenticate` header of cli-response. WWW-Authenticate: Basic realm="$realm"

auth-type

The type of the authentication scheme, currently only supports basic

auth-secret

The secret refs of the username and password, format is ns/name

auth-secret-type

Secret supports two types:

1. auth-file: secret's data only contains one key "auth", and its value is the string of Apache htpasswd format. for example:

```
data:
  auth: "user1:$apr1$xyz..."
```

2. auth-map: secret's data each key represents a username, and the corresponding value is the password hash (without the username in htpasswd format). for example:

```
data:
  user1: "$apr1$xyz...."
  user2: "$apr1$abc...."
```

Note: Currently, only htpasswd format password hashes generated using the apr1 algorithm are supported.

CR

ALB CR has added auth-related configuration items that can be configured on ALB/Frontend/Rule CRs. During runtime, ALB will convert the annotations on Ingress into rules.

```
auth:
  # Basic authentication configuration
  basic:
    #  string; corresponding to nginx.ingress.kubernetes.io/auth-type: ba
sic
    auth_type: 'basic'
    #  string; corresponding to nginx.ingress.kubernetes.io/auth-realm
    realm: 'Restricted Access'
    #  string; corresponding to nginx.ingress.kubernetes.io/auth-secret
    secret: 'ns/name'
    #  string; corresponding to nginx.ingress.kubernetes.io/auth-secret-t
ype
    secret_type: 'auth-map|auth-file'
  # Forward authentication configuration
  forward:
    #  boolean; corresponding to nginx.ingress.kubernetes.io/auth-always-
set-cookie
    always_set_cookie: true
    #  string; corresponding to nginx.ingress.kubernetes.io/auth-proxy-se
t-headers
    auth_headers_cm_ref: 'ns/name'
    #  string; corresponding to nginx.ingress.kubernetes.io/auth-request-
redirect
    auth_request_redirect: '/login'
    #  string; corresponding to nginx.ingress.kubernetes.io/auth-method
    method: 'GET'
    #  string; corresponding to nginx.ingress.kubernetes.io/auth-signin
    signin: '/signin'
    #  string; corresponding to nginx.ingress.kubernetes.io/auth-signin-r
edirect-param
    signin_redirect_param: 'redirect_to'
    #  []string; corresponding to nginx.ingress.kubernetes.io/auth-respon
se-headers
    upstream_headers:
      - 'X-User-ID'
      - 'X-User-Name'
      - 'X-User-Email'
    #  string; corresponding to nginx.ingress.kubernetes.io/auth-url
    url: 'http://auth-service/validate'
```

Auth supports configuration on:

- Alb CR's `.spec.config.auth`
- Frontend CR's `.spec.config.auth`
- Rule CR's `.spec.config.auth`

The inheritance order is Alb > Frontend > Rule. If a child cr is not configured, the configuration of the parent cr will be used.

ALB Special Ingress Annotation

In the process of handling Ingress, ALB determines the priority based on the prefix of the annotation. The priority from high to low is:

- `index.$rule_index-$path_index.alb.ingress.cpaas.io`
- `alb.ingress.cpaas.io`
- `nginx.ingress.kubernetes.io`

This can handle the compatibility problem with ingress-nginx and specify the auth configuration on a specific Ingress path.

Auth-Enable

```
alb.ingress.cpaas.io/auth-enable: 'false'
```

A new annotation added by ALB, used to specify whether to enable authentication functionality for the Ingress.

Ingress-Nginx Auth Related Other Features

Global-Auth

In ingress-nginx, you can set a global auth through the ConfigMap. This is equivalent to configuring auth for all Ingresses. In ALB, you can configure auth on the ALB2 and FT CRs.

The rules under them will inherit these configurations.

No-Auth-Locations

In ALB, you can disable the auth function of this Ingress by configuring the annotation: `alb.ingress.cpaas.io/auth-enable: "false"` on the Ingress.

Note: Incompatible Parts with Ingress-Nginx

1. Does not support auth-keepalive
2. Does not support auth-snippet
3. Does not support auth-cache
4. Does not support auth-tls
5. Basic-auth only supports basic, does not support digest
6. Basic-auth basic only supports apr1 algorithm, does not support bcrypt sha256, etc.

Troubleshooting

1. Check ALB pod Nginx container log
2. Check the `X-ALB-ERR-REASON` header in the return

Trademark Notice

NGINX® is a registered trademark of F5, Inc. in the U.S. and in other countries.

Alauda is an independent vendor. This product is not affiliated with, endorsed by, or sponsored by F5, Inc. All trademarks are the property of their respective owners and are used here for identification purposes only.

Deploy High Available VIP for ALB

The high availability of the **ALB** requires a VIP. There are two ways to obtain a VIP.

TOC

Method 1: Use LoadBalancer type Service to provide VIP

Method 2: Use external ALB device to provide VIP

Trademark Notice

Method 1: Use LoadBalancer type Service to provide VIP

When creating an ALB in `container` network mode, the system automatically creates a LoadBalancer Service to provide a VIP for that ALB.

Before using this, ensure the cluster supports LoadBalancer Services. You can use the platform built-in implementation. For setup, see Configure MetalLB.

After MetalLB is ready, you can add the following annotations to `alb.spec.config.vip.lbSvcAnnotations` to tweak MetalLB behavior. See ALB networking configuration.

Annotation	Description
`metallb.universe.tf/loadBalancerIPs`	Comma-separated VIPs to allocate to the Service. Use multiple entries for multi-VIP or dual-stack, for example: `192.0.2.10,2001:db8::10`.
`metallb.universe.tf/address-pool`	The MetalLB address pool to allocate from.

Method 2: Use external ALB device to provide VIP

- Please confirm with the network engineer the IP address (public IP, private IP, VIP) or domain name of the ALB service before deployment. If you want to use a domain name as the address for external traffic to access the ALB, you need to apply for a domain name in advance and configure domain name resolution. It is recommended to use a commercial Load Balancer device to provide a VIP, if not, you can use the Pure Software Data Center LB Solution (Alpha)
- According to the business scenario, the external ALB needs to configure health checks for all the ports in use to reduce the downtime of ALB upgrade. The health check configuration is as follows:

Health Check Parameters	Description
Port	• For global clusters, fill in: 11782. • For workload clusters, fill in: 1936.
Protocol	The protocol type of the health check, it is recommended to use TCP.
Response Timeout	The time required to receive the health check response, it is recommended to configure it to 2 seconds.
Check Interval	The time interval for the health check, it is recommended to configure it to 5 seconds.

Health Check Parameters	Description
Unhealthy Threshold	The number of consecutive failures after which the health check status of the backend server is determined to be failed, it is recommended to configure it to 3 times.

Trademark Notice

MetalLB™ is a trademark of The Linux Foundation.

Alauda is an independent vendor. This product is not affiliated with, endorsed by, or sponsored by The Linux Foundation. All trademarks are the property of their respective owners and are used here for identification purposes only.

Decision-Making for ALB Performance Selection

For the platform's proposed specifications for **small**, **medium**, **large**, and **custom** production environments, as well as the resource allocation methods for **instances** and **ports**, the following suggestions can be referenced for deployment.

TOC

Small Production Environment

Medium Production Environment

Large Production Environment

Special Scenario Deployment Recommendations

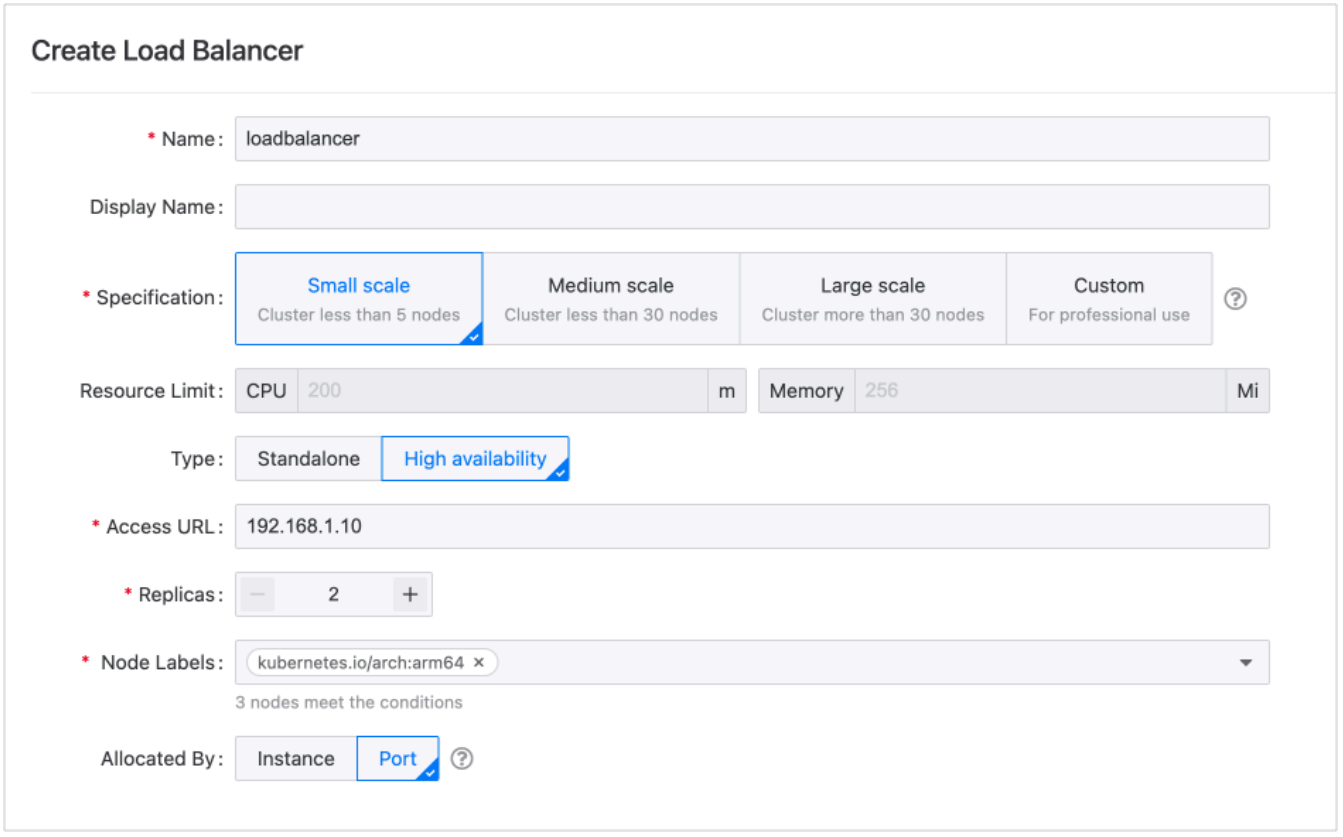
Load Balancer Usage Mode Selection

Small Production Environment

For smaller business scales, such as having no more than 5 nodes in the cluster and only used for running standard applications, a **single** load balancer is sufficient. It is recommended to use it in a **high availability** mode with at least 2 replicas to ensure stability in the environment.

You can isolate the load balancer using **port** isolation, allowing multiple projects to share it.

The peak QPS measured in a lab environment for this specification is approximately 300 requests per second.

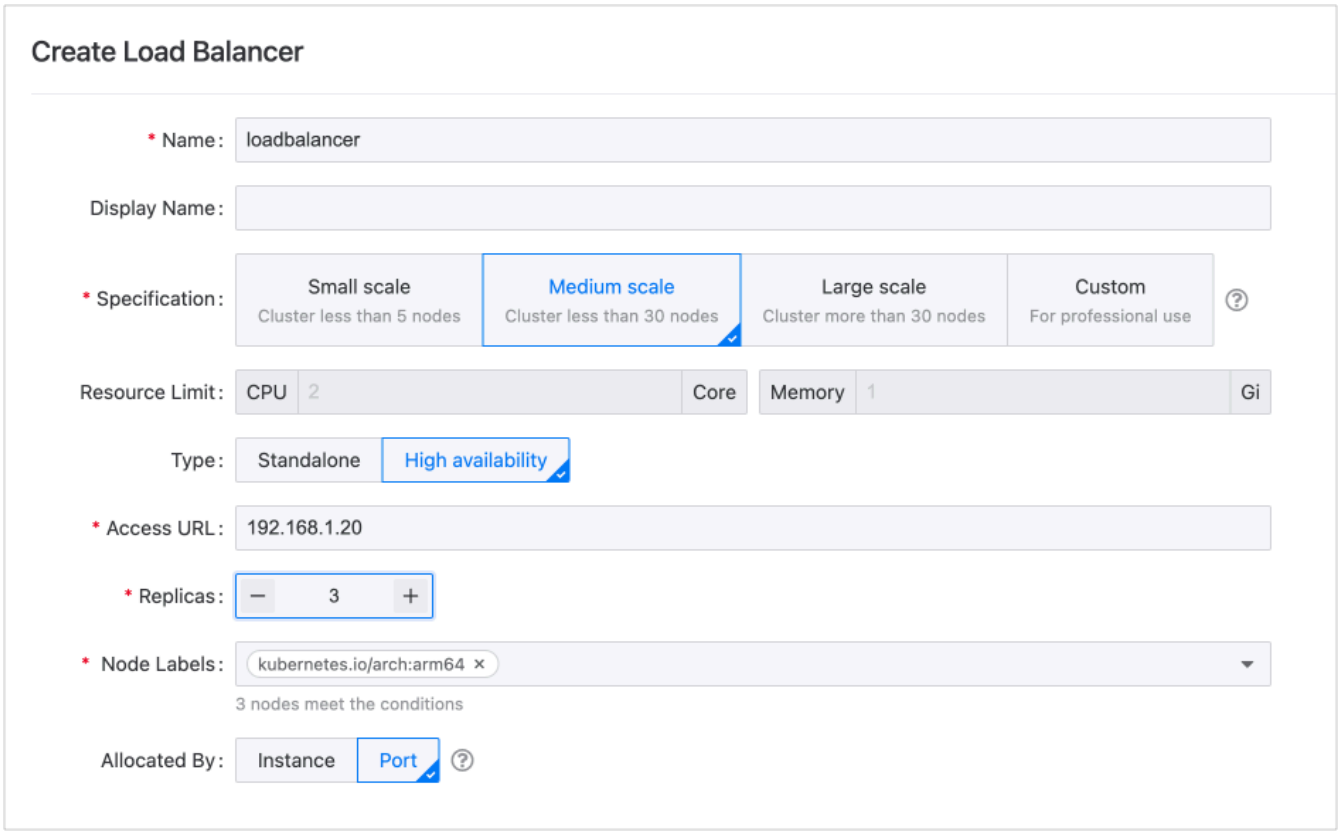


Medium Production Environment

When the business volume reaches a certain scale, such as having no more than 30 nodes in the cluster and needing to handle high-concurrency business alongside running standard applications, a **single** load balancer will still be adequate. It is advisable to employ a **high availability** mode with at least 3 replicas to maintain stability in the environment.

You can utilize either **port** isolation or **instance** allocation methods to share the load balancer among multiple projects. Of course, you can also create new load balancers for dedicated use by core projects.

The peak QPS measured in a lab environment for this specification is around 10,000 requests per second.

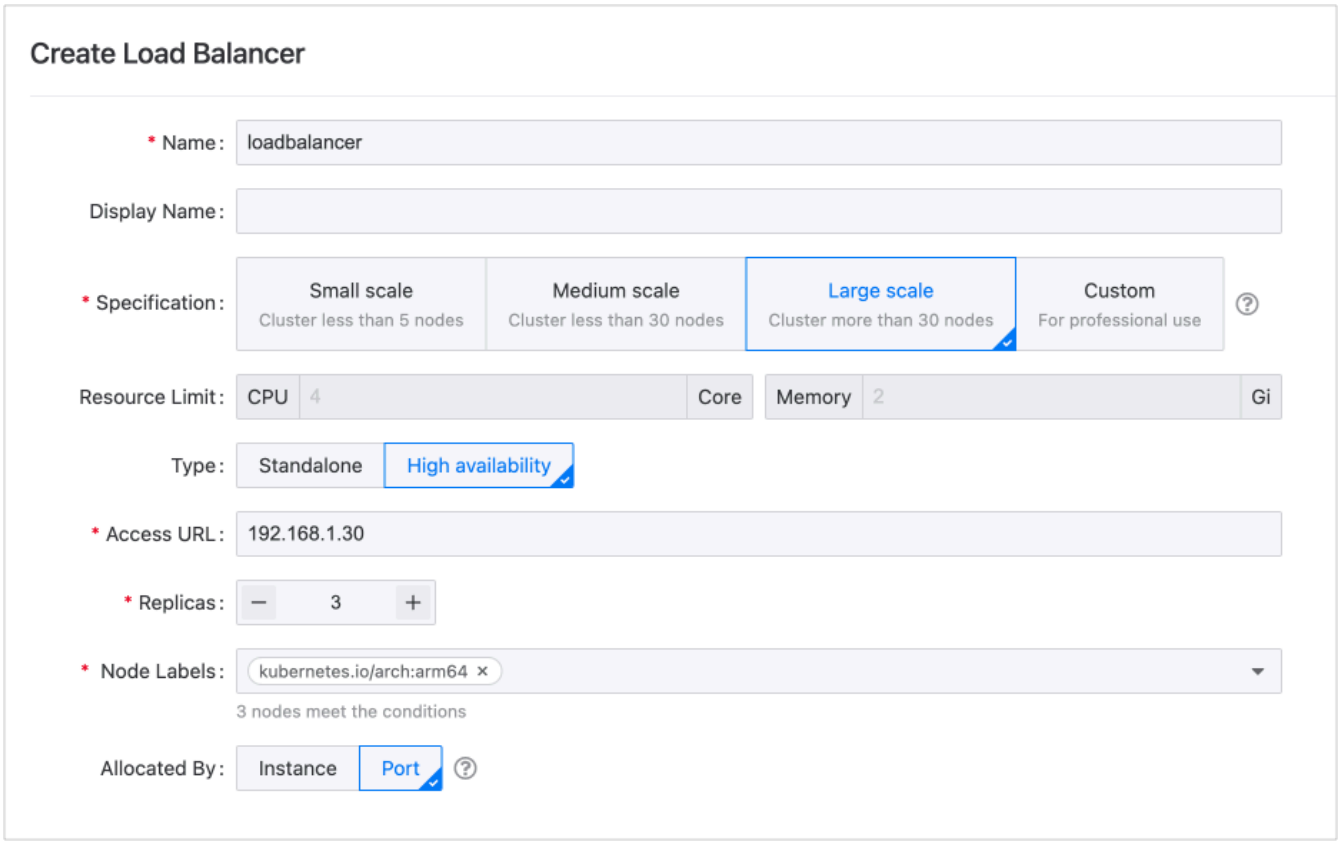


Large Production Environment

For larger business volumes, such as having more than 30 nodes in the cluster and needing to handle high-concurrency business as well as long-lived data connections, it is recommended to use **multiple** load balancers, each in a **high availability** type with at least 3 replicas to ensure stability in the environment.

You can isolate the load balancer using either **port** isolation or **instance** allocation methods for multiple projects to share it. You may also create new load balancers for exclusive use by core projects.

The peak QPS measured in a lab environment for this specification is approximately 20,000 requests per second.



Special Scenario Deployment Recommendations

Scenario	Deployment Recommendations
Function Testing	It is advisable to deploy a **single instance** of the load balancer.
Testing Environment	If the testing environment meets the definitions of **small** or **medium** as stated above, using a **single point** load balancer is sufficient. The load balancer **instance** can be shared among **multiple projects**.
Core Applications	It is recommended to use specific load balancers exclusively for core applications.
Transferring Large Scale Data	Due to minimal memory consumption caused by the load balancer itself, it is sufficient to reserve 2Gi of memory even for the **large** specification. However, if the business requires transferring large-scale data, which will lead to substantial memory consumption, the memory allocation for the load balancer should be increased accordingly. It is recommended to gradually expand the memory of the load balancer in **custom** specification scenarios, closely monitoring memory usage to ultimately arrive at an acceptable memory size for reasonable usage rates.

Load Balancer Usage Mode Selection

Usage Mode	Advantages	Disadvantages
(Recommended) Allocate the load balancer as an instance resource to a single project	• Management is relatively simple. • Each project has its own load balancer, ensuring rule isolation and resource separation, with no interference.	In host network mode, the cluster must possess a significant number of nodes available for the load balancer, resulting in high resource consumption requirements.
Allocate the load balancer as an instance resource to multiple projects	Management is relatively straightforward.	Since all assigned projects hold full permissions for the load balancer instance, when one project configures the ports and rules of the load balancer, the following situations may arise: • The rules configured by that project may affect other projects. • Mis-operations during load balancer configuration might alter other projects' settings. • Traffic requests from a particular business may impact the overall availability of the load balancer instance.

Usage Mode	Advantages	Disadvantages
Dynamically allocate load balancer resources by port, with different projects using different ports	The rules between projects isolate them, ensuring no interference.	• Management complexity increases. Platform administrators must actively plan and allocate ports for projects and configure external service mappings. • The maturity of port-based allocation is lower. Currently, it is used by fewer clients and requires further refining of features. • Resource conflicts. All services using the same load balancer may face scenarios where a single service negatively impacts the entire load balancer.

Load Balancing Session Affinity Policy in ALB

This guide explains the various load balancing algorithms available in ALB and how to configure them to optimize traffic distribution across your application pods.

TOC

Overview
Available Algorithms
Round Robin (Default)
Source IP Hash
Cookie-Based Affinity
Header-Based Affinity
EWMA (Exponentially Weighted Moving Average)
Configuration Methods
1. Using Ingress Annotations
2. Using ALB Frontend/Rule Custom Resource
Best Practices

Overview

ALB supports multiple load balancing algorithms to distribute incoming traffic across backend pods. The choice of algorithm depends on your application requirements, such as session persistence, performance optimization, or even distribution of load.

Available Algorithms

Round Robin (Default)

- **Policy**: `rr`
- **Description**: Distributes requests sequentially across all available pods in a circular order.
- **Use Case**: Best for stateless applications where each request can be handled by any pod.

Source IP Hash

- **Policy**: `sip-hash`
- **Description**: Consistently routes requests from the same source IP address to the same pod.
- **Behavior**: Uses the first IP in X-Forwarded-For header if present, otherwise uses the client's source IP.
- **Use Case**: Useful when client IP-based session persistence is required.

Cookie-Based Affinity

- **Policy**: `cookie`
- **Attribute**: `cookie-name` (defaults to `JSESSIONID`)
- **Description**: Routes requests with the same cookie value to the same pod.
- **Behavior**:
  - If the specified cookie is not present, ALB adds it to the response
  - Cookie format: `timestamp.worker_pid.random_number`
- **Use Case**: Ideal for applications requiring session stickiness based on cookies.

Header-Based Affinity

- **Policy**: `header`
- **Attribute**: `header-name`
- **Description**: Routes requests with the same header value to the same pod.
- **Use Case**: Suitable for applications that need routing based on specific HTTP headers.

EWMA (Exponentially Weighted Moving Average)

- **Policy**: `ewma`
- **Description**: Routes traffic based on pod response times using the Power of Two Choices (P2C) algorithm with EWMA.
- **Behavior**:
  - Maintains an EWMA score for each pod based on response times
  - Routes traffic to pods with lower EWMA scores
  - Scores decay exponentially over time
- **Use Case**: Optimal for latency-sensitive applications
- **Reference**: Twitter Finagle EWMA Documentation ↗

Configuration Methods

1. Using Ingress Annotations

```
annotations:
  alb.ingress.cpaas.io/session-affinity-policy: '<algorithm>' # rr | sip-hash | cookie | header | ewma
  alb.ingress.cpaas.io/session-affinity-attribute: '<attribute>' # Required for cookie and header policies
```

2. Using ALB Frontend/Rule Custom Resource

```
spec:
  serviceGroup:
    session_affinity_policy: '<algorithm>' # rr | sip-hash | cookie | hea
der | ewma
    session_affinity_attribute: '<attribute>' # Required for cookie and h
eader policies
```

Best Practices

- Choose Round Robin for stateless applications with uniform request patterns
- Use Source IP Hash when client IP-based session persistence is required
- Implement Cookie-based affinity for web applications requiring session stickiness
- Consider EWMA for services with varying response times to optimize latency

L4/L7 Timeout

TOC

Basic Concepts
CRD
What Timeout Means
Ingress Annotation
Port Level Timeout
Trademark Notice

Basic Concepts

L4/L7 timeout is a feature provided by ALB. It is used to configure the timeout time for L4/L7 proxy.

Timeout is implemented through a Lua script, and Nginx **does not need to reload** when it is changed.

CRD

```
timeout:
  properties:
    proxy_connect_timeout_ms:
      type: integer
    proxy_read_timeout_ms:
      type: integer
    proxy_send_timeout_ms:
      type: integer
  type: object
```

Config can be configured on:

- Frontend: `.spec.config.timeout`
- Rule: `.spec.config.timeout`

What Timeout Means

There are three types of timeouts:

1. **proxy_connect_timeout_ms**: Defines the timeout for establishing a connection with the upstream server. If the connection cannot be established within this time, the request will fail.
2. **proxy_read_timeout_ms**: Defines the timeout for reading a response from the upstream server. The timeout is set between two successive read operations, not for the entire response. If no data is received within this time, the connection is closed.
3. **proxy_send_timeout_ms**: Defines the timeout for sending a request to the upstream server. Similar to the read timeout, this is set between two successive write operations. If no data can be sent within this time, the connection is closed.

Ingress Annotation

Annotation	Description
nginx.ingress.kubernetes.io/proxy-connect-timeout	Corresponds to proxy_connect_timeout_ms in CR
nginx.ingress.kubernetes.io/proxy-read-timeout	Corresponds to proxy_read_timeout_ms in CR
nginx.ingress.kubernetes.io/proxy-send-timeout	Corresponds to proxy_send_timeout_ms in CR

Port Level Timeout

You can configure timeout on a port directly, which is used as an L4 timeout.

Trademark Notice

NGINX® is a registered trademark of F5, Inc. in the U.S. and in other countries.

Alauda is an independent vendor. This product is not affiliated with, endorsed by, or sponsored by F5, Inc. All trademarks are the property of their respective owners and are used here for identification purposes only.

HTTP Redirect

TOC

Basic Concepts
CRD
Ingress Annotation
SSL-Redirect
Port-Level Redirect
Rule-Level Redirect
Trademark Notice

Basic Concepts

HTTP redirect is a feature provided by ALB. It will directly return a 30x HTTP code for the request that matches the rule. The Location header will be used to instruct the client to redirect to the new URL.

ALB supports redirect configuration at the port and rule levels.

CRD

```
redirect:
  properties:
    code:
      type: integer
    host:
      type: string
    port:
      type: integer
    prefix_match:
      type: string
    replace_prefix:
      type: string
    scheme:
      type: string
    url:
      type: string
  type: object
```

Redirect could be configured on:

- Frontend: `.spec.config.redirect`
- Rule: `.spec.config.redirect`

Ingress Annotation

Annotation	Description
nginx.ingress.kubernetes.io/permanent-redirect	Corresponds to URL in CR, will set code to 301 by default
nginx.ingress.kubernetes.io/permanent-redirect-code	Corresponds to code in CR
nginx.ingress.kubernetes.io/temporal-redirect	Corresponds to URL in CR, will set code to 302 by default
nginx.ingress.kubernetes.io/temporal-redirect-code	Corresponds to code in CR

Annotation	Description
nginx.ingress.kubernetes.io/ssl-redirect	Corresponds to scheme in CR, will set scheme to HTTPS by default
nginx.ingress.kubernetes.io/force-ssl-redirect	Corresponds to scheme in CR, will set scheme to HTTPS by default

SSL-Redirect

1. SSL-redirect and force-ssl-redirect differ in that SSL-redirect only takes effect when the ingress has a certificate for the corresponding domain, while force-ssl-redirect takes effect regardless of whether there is a certificate.
2. For HTTPS ports, if only SSL-redirect is configured, the redirect will not be set.

Port-Level Redirect

When redirect is configured at the port level, *all requests* to this port will be redirected according to the redirect configuration.

Rule-Level Redirect

When redirect is configured at the rule level, the request matching this rule will be redirected according to the redirect configuration.

Trademark Notice

NGINX® is a registered trademark of F5, Inc. in the U.S. and in other countries.

Alauda is an independent vendor. This product is not affiliated with, endorsed by, or sponsored by F5, Inc. All trademarks are the property of their respective owners and are used here for identification purposes only.

CORS

TOC

Basic Concepts

CRD

Basic Concepts

CORS (Cross-Origin Resource Sharing) is a mechanism that allows resources (e.g., fonts, JavaScript, etc.) on a web page to be requested from another domain outside the domain from which the resource originated.

CRD

```
enableCORS:
  description: enableCORS is the switch whether enable cross domain,
    when EnableCORS is false, alb2 transports information to backend
    servers which determine whether allow cross-domain
  type: boolean
corsAllowHeaders:
  description: corsAllowHeaders defines the headers allowed by cors
    when enableCORS is true, multiple headers are separated by commas
  type: string
corsAllowOrigin:
  description: corsAllowOrigin defines the origin allowed by cors when
    enableCORS is true, multiple origins are separated by commas
  type: string
```

It can be configured on the rule `.spec`.

Header Modification

TOC

Basic Concepts
use annotations
examples
Trademark Notice

Basic Concepts

When a request is received, header modification allows adjusting request headers before forwarding to the backend. Similarly, when a response is received, it allows adjusting response headers before they are returned to the client.

use annotations

target	annotation key
ingress	`alb.ingress.cpaas.io/rewrite-request`, `alb.ingress.cpaas.io/rewrite-response`
rule	`alb.rule.cpaas.io/rewrite-request`, `alb.rule.cpaas.io/rewrite-response`

Annotation values are JSON strings containing the config.

```
type RewriteRequestConfig struct {
    Headers       map[string]string   `json:"headers,omitempty"`
// set header
    HeadersVar    map[string]string   `json:"headers_var,omitempty"`
// set header, which values are variable name
    HeadersRemove []string           `json:"headers_remove,omitempty"`
// remove header
    HeadersAdd    map[string][]string `json:"headers_add,omitempty"`
// add header, which values could be multiple
    HeadersAddVar map[string][]string `json:"headers_add_var,omitempty"`
// add header, which values could be multiple and which values are variab
le name
}
type RewriteResponseConfig struct {
    Headers       map[string]string   `json:"headers,omitempty"`
// set header
    HeadersRemove []string           `json:"headers_remove,omitempty"`
// remove header
    HeadersAdd    map[string][]string `json:"headers_add,omitempty"`
// add header, which values could be multiple
}
```

Note: in `*_var` maps the key is the header name and the value is the ALB context variable name. For example, add the following annotation to an Ingress:

```
alb.ingress.cpaas.io/rewrite-request: '{
    "headers_var": {
        "x-my-host": "http_host"
    }
}'
```

will add key `x-my-host` and value of request host header to request header. you can refer to the nginx variable ↗ for the variable names.

ALB provides additional variables:

variable name	description
`first_forward_or_remote_addr`	the first forwarded address or the remote address, default is `remote_addr`
`first_forward`	the first forwarded address , default is empty string

examples

To add an Authorization header from a cookie, you can use:

```
alb.ingress.cpaas.io/rewrite-request: '{"headers_var":{"Authorization":"c
ookie_auth_token"}}'
```

To set HSTS, you can use:

```
alb.rule.cpaas.io/rewrite-response: |
  { "headers": { "Strict-Transport-Security": "max-age=63072000; includeS
ubDomains; preload"} }
```

Trademark Notice

NGINX® is a registered trademark of F5, Inc. in the U.S. and in other countries.

Alauda is an independent vendor. This product is not affiliated with, endorsed by, or sponsored by F5, Inc. All trademarks are the property of their respective owners and are used here for identification purposes only.

URL Rewrite

TOC

Basic Concepts

Configuration

Trademark Notice

Basic Concepts

ALB can rewrite the request URL before forwarding it to the backend. You can use regex capture groups to rewrite the URL.

Configuration

via ingress annotation

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  nginx.ingress.kubernetes.io/rewrite-target: /$2
  name: demo
spec:
  ingressClassName: alb
  rules:
    - http:
        paths:
          - backend:
              service:
                name: frontend
                port:
                  number: 8080
            path: /(prefix-x)(/|$)(.*)
            pathType: ImplementationSpecific
```

via rule

```yaml
apiVersion: crd.alauda.io/v1
kind: Rule
metadata:
  labels:
    alb2.cpaas.io/frontend: alb-00080
    alb2.cpaas.io/name: alb
  name: demo
  namespace: cpaas-system
spec:
  dslx:
    - type: URL
      values:
        - - REGEX
          - ^/(prefix-x)(/|$)(.*)
  rewrite_base: /(prefix-x)(/|$)(.*)
  rewrite_target: /$3
```

Example: client requests `/prefix-x/abc`; backend receives `/abc`.

Trademark Notice

NGINX® is a registered trademark of F5, Inc. in the U.S. and in other countries.

Alauda is an independent vendor. This product is not affiliated with, endorsed by, or sponsored by F5, Inc. All trademarks are the property of their respective owners and are used here for identification purposes only.

ModSecurity

ModSecurity is an open-source Web Application Firewall (WAF) designed to protect web applications from malicious attacks. It is maintained by the open-source community and supports various programming languages and web servers. The platform Load Balancer (ALB) supports configuring ModSecurity, allowing for individual configurations at the Ingress level.

TOC

Terminology
Procedure to Operate
Method One: Add Annotations
Method Two: Configure CR
Related Explanations
Override
Configuration Example
Trademark Notice

Terminology

Term	Explanation
owasp-core-rules	The OWASP Core Rule Set is an open-source ruleset used to detect and prevent common web application attacks.

Procedure to Operate

Configure ModSecurity by adding annotations to the corresponding resource's YAML file or by configuring CR.

Method One: Add Annotations

Add the following annotations to the metadata.annotations field of the corresponding YAML file to configure ModSecurity.

- **Ingress-Nginx Compatible Annotations**

Annotation	Type	Applicable Object	Explanati
nginx.ingress.kubernetes.io/enable-modsecurity	bool	Ingress	Enable ModSecur
nginx.ingress.kubernetes.io/enable-owasp-core-rules	bool	Ingress	Enable the OWASP C Rule Set.
nginx.ingress.kubernetes.io/modsecurity-transaction-id	string	Ingress	Used to identify un transactior for each request, ai in logging debugging

Annotation	Type	Applicable Object	Explanati
nginx.ingress.kubernetes.io/modsecurity-snippet	string	Ingress, ALB, FT, Rule	Allows use insert cust ModSecur configurati to meet specific security requireme

- **ALB Special Annotations**

Annotation	Type	Applicable Object	Explanation
alb.modsecurity.cpaas.io/use-recommend	bool	Ingress	Enable or disable recommended ModSecurity rules; se `true` to apply a predefined set of secu rules.
alb.modsecurity.cpaas.io/cmref	string	Ingress	Reference specific configurations, e.g., custom security configurations can be loaded by specifying t ConfigMap's referenc path (`$ns/$name#$sectio`

Method Two: Configure CR

1. Open the ALB, FT, or Rule configuration file that needs to be configured.

2. Add the following fields under spec.config as required.

```
{ 'modsecurity': {
      'enable': true, # Enable or disable ModSecurity
      'transactionId': '$xx', # Use ID from Nginx
      'useCoreRules': true, # Add modsecurity_rules_file /etc/nginx/owasp-modsecurity-crs/nginx-modsecurity.conf
      'useRecommend': true, # Add modsecurity_rules_file /etc/nginx/modsecurity/modsecurity.conf
      'cmRef': '$ns/$name#$section', # Add configuration from ConfigMap
    } }
```

3. Save and apply the configuration file.

Related Explanations

Override

If ModSecurity is not configured in the Rule, it will attempt to find the configuration in FT; if there is no configuration in FT, it will use the configuration from ALB.

Configuration Example

The following example deploys an ALB named `waf-alb` and a demo backend application named `hello`. Additionally, an Ingress named `ing-waf-enable` is deployed, which defines the `/waf-enable` route and configures ModSecurity rules. Any request containing the query parameter `test`, where the value includes the string `test`, will be blocked.

```
cat <<EOF | kubectl apply -f -
apiVersion: crd.alauda.io/v2
kind: ALB2
metadata:
  name: waf-alb
  namespace: cpaas-system
spec:
  config:
    loadbalancerName: waf-alb
    projects:
      - ALL_ALL
    replicas: 1
  type: nginx
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    nginx.ingress.kubernetes.io/enable-modsecurity: "true"
    nginx.ingress.kubernetes.io/modsecurity-transaction-id: "$request_id"
    nginx.ingress.kubernetes.io/modsecurity-snippet: |
      SecRuleEngine On
      SecRule ARGS:test "@contains test" "id:1234,deny,log"
  name: ing-waf-enable
spec:
  ingressClassName: waf-alb
  rules:
    - http:
        paths:
          - backend:
              service:
                name: hello
                port:
                  number: 80
            path: /waf-enable
            pathType: ImplementationSpecific
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ing-waf-normal
spec:
  ingressClassName: waf-alb
```

```
  rules:
    - http:
        paths:
          - backend:
              service:
                name: hello
                port:
                  number: 80
            path: /waf-not-enable
            pathType: ImplementationSpecific
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello
spec:
  replicas: 1
  selector:
    matchLabels:
      service.cpaas.io/name: hello
      service_name: hello
  template:
    metadata:
      labels:
        service.cpaas.io/name: hello
        service_name: hello
    spec:
      containers:
        - name: hello-world
          image: hashicorp/http-echo
          imagePullPolicy: IfNotPresent
---
apiVersion: v1
kind: Service
metadata:
  name: hello
spec:
  internalTrafficPolicy: Cluster
  ipFamilies:
    - IPv4
  ipFamilyPolicy: SingleStack
  ports:
    - name: http
      port: 80
```

```
      protocol: TCP
      targetPort: 5678
  selector:
    service_name: hello
  sessionAffinity: None
  type: ClusterIP
EOF
```

Trademark Notice

NGINX® is a registered trademark of F5, Inc. in the U.S. and in other countries.

Alauda is an independent vendor. This product is not affiliated with, endorsed by, or sponsored by F5, Inc. All trademarks are the property of their respective owners and are used here for identification purposes only.

OTel

OpenTelemetry (OTel) is an open-source project aimed at providing a vendor-neutral standard for collecting, processing, and exporting telemetry data in distributed systems, such as microservices architectures. It helps developers analyze the performance and behavior of software more easily, thus facilitating the diagnosis and resolution of application issues.

TOC

- Terminology
- Prerequisites
- Procedure
  - Update ALB Configuration
- Related Operations
  - Configuring OTel in Ingress
  - Using OTel in Applications
  - Inheritance
- Additional Notes
  - Sampling Strategies
  - Attributes
- Configuration Example
- Trademark Notice

Terminology

Term	Explanation
Trace	The data submitted to the OTel Server, which is a collection of related events or operations used to track the flow of requests in distributed systems; each Trace consists of multiple Spans.
Span	An independent operation or event within a Trace that includes start time, duration, and other relevant information.
OTel Server	An OTel server capable of receiving and storing Trace data, such as Jaeger, Prometheus, etc.
Jaeger	An open-source distributed tracing system used for monitoring and troubleshooting microservices architectures, supporting integration with OpenTelemetry.
Attributes	Key-value pairs attached to a Trace or Span to provide additional contextual information. This includes Resource Attributes and Span Attributes; see Attributes for more information.
Sampler	A strategy component that determines whether to sample and report a Trace. Different sampling strategies can be configured, such as full sampling, proportional sampling, etc.
ALB (Another Load Balancer)	A software or hardware device that distributes network requests across available nodes in a cluster; the load balancer (ALB) used in the platform is a layer 7 software load balancer, which can be configured to monitor traffic with OTel. ALB supports submitting Traces to a specified Collector and allows different sampling strategies; it also supports configuring whether to submit Traces at the Ingress level.
FT (Frontend)	The port configuration for ALB, specifying port-level configurations.
Rule	Routing rules on the port (FT) used to match specific routes.
HotROD (Rides on Demand)	A sample application provided by Jaeger to demonstrate the use of distributed tracing; refer to Hot R.O.D. - Rides on Demand ↗ for more details.

Term	Explanation
hotrod-with-proxy	Specifies the addresses of HotROD's internal microservices via environment variables; refer to hotrod-with-proxy ↗ for more details.

Prerequisites

- **Ensure that an operable ALB exists**: Create or use an existing ALB. The examples below use `<otel-alb>` as the ALB name. For instructions on creating an ALB, see Configure ALB.
- **Ensure that there is an OTel data reporting server address**: This address will hereinafter be referred to as `<jaeger-server>`.

Procedure

Update ALB Configuration

1. On a control plane node of the cluster, use the CLI tool to execute the following command to edit the ALB configuration.

   ```
   kubectl edit alb2 -n cpaas-system <otel-alb> # Replace <otel-alb> with the actual ALB name
   ```

2. Add the following fields under the `spec.config` section.

   ```
   otel:
     enable: true
     exporter:
       collector:
         address: '<jaeger-server>' # Replace <jaeger-server> with the actual OTel data reporting server address
         request_timeout: 1000
   ```

   Example configuration once completed:

```
spec:
  address: 192.168.1.1
  config:
    otel:
     enable: true
     exporter:
       collector:
         address: "http://jaeger.default.svc.cluster.local:4318"
         request_timeout: 1000
    antiAffinityKey: system
    defaultSSLCert: cpaas-system/cpaas-system
    defaultSSLStrategy: Both
    gateway:
    ...
type: nginx
```

3. Execute the following command to save the updates. After the update, the ALB will default to enabling OpenTelemetry, and all request Trace information will be reported to the Jaeger Server.

```
:wq
```

Related Operations

Configuring OTel in Ingress

- **Enable or Disable OTel on Ingress**
  By configuring whether to enable OTel on Ingress, you can better monitor and debug the request flow of applications, identifying performance bottlenecks or errors by tracing requests as they propagate between different services.
  Procedure
  Add the following configuration under the metadata.annotations field of Ingress:

```
nginx.ingress.kubernetes.io/enable-opentelemetry: "true"
```

  Parameter Explanation:

- **nginx.ingress.kubernetes.io/enable-opentelemetry**: When set to `true`, it indicates that the Ingress controller enables OpenTelemetry functionality while processing requests through this Ingress, which means request Trace information will be collected and reported. When set to `false` or this annotation is removed, it means that request Trace information will not be collected or reported.

- **Enable or Disable OTel Trust on Ingress**

  OTel Trust determines whether Ingress trusts and uses the Trace information (e.g., trace ID) from incoming requests.

  Procedure

  Add the following configuration under the metadata.annotations field of Ingress:

  ```
  nginx.ingress.kubernetes.io/opentelemetry-trust-incoming-span: "true"
  ```

  Parameter Explanation:

  - **nginx.ingress.kubernetes.io/opentelemetry-trust-incoming-span**: When set to `true`, the Ingress continues to use already existing Trace information, helping maintain consistency in cross-service tracing, allowing the entire request chain to be fully traced and analyzed in the distributed tracing system. When set to `false`, it will generate new tracing information for the request, which may cause the request to be treated as part of a new tracing chain after entering the Ingress, interrupting cross-service trace continuity.

- **Add Different OTel Configurations on Ingress**

  This configuration allows you to customize OTel's behavior and data export methodology for different Ingress resources, enabling fine-grained control over each service's tracing strategy or target.

  Procedure

  Add the following configuration under the metadata.annotations field of Ingress:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    alb.ingress.cpaas.io/otel: >
      {
         "enable": true,
         "exporter": {
             "collector": {
                 "address": "<jaeger-server>", # Replace <jaeger-server
> with the actual OTel data reporting server address, e.g., "address":
"http://128.0.0.1:4318"
                 "request_timeout": 1000
             }
         }
      }
```

Parameter Explanation:

- **exporter**: Specifies how the collected Trace data is sent to the OTel Collector (the OTel data reporting server).
- **address**: Specifies the address of the OTel Collector.
- **request_timeout**: Specifies the request timeout.

Using OTel in Applications

The following configuration shows the complete OTel configuration structure, which can be used to define how to enable and use OTel features in applications.

On a control plane node of the cluster, use the CLI tool to execute the following command to get the complete OTel configuration structure.

```
kubectl get crd alaudaloadbalancer2.crd.alauda.io -o json|jq ".spec.versi
ons[2].schema.openAPIV3Schema.properties.spec.properties.config.propertie
s.otel"
```

Echoed Result:

```
{
    "otel": {
        "enable": true
    }
    "exporter": {
        "collector": {
            "address": ""
          },
    },
    "flags": {
        "hide_upstream_attrs": false
        "notrust_incoming_span": false
        "report_http_request_header": false
        "report_http_response_header": false
    },
    "sampler": {
        "name": "",
        "options": {
            "fraction": ""
            "parent_name": ""
          },
      },
 }
```

Parameter Explanation:

Parameter	Description
otel.enable	Whether to enable OTel functionality.
exporter.collector.address	The address of the OTel data reporting server, supporting http/https protocols and domain names.
flags.hide_upstream_attrs	Whether to report information about upstream rules.
flag.notrust_incoming_span	Whether to trust and use the OTel Trace information (e.g., trace ID) from incoming requests.
flags.report_http_request_header	Whether to report request headers.
flags.report_http_response_header	Whether to report response headers.

Parameter	Description
sampler.name	Sampling strategy name; see Sampling Strategies for details.
sampler.options.fraction	Sampling rate.
sampler.options.parent_name	The parent strategy for parent_base sampling strategies.

Inheritance

By default, if the ALB configures certain OTel parameters and FT is not configured, FT will inherit the parameters from the ALB as its own configuration; that is, FT inherits the ALB configuration, while Rule can inherit configurations from both ALB and FT.

- **ALB**: The configuration on the ALB is typically global and default. You can configure global parameters such as Collector addresses here, which will be inherited by the lower-level FT and Rule.
- **FT**: FT can inherit configurations from ALB, meaning that certain OTel parameters that are not configured on FT will use the configuration from ALB. However, FT can also be refined further; for instance, you can choose to selectively enable or disable OTel on FT without affecting other FT or the global settings of ALB.
- **Rule**: Rule can inherit configurations from both ALB and FT. However, Rule can also be refined further; for instance, a specific Rule can choose not to trust the incoming OTel Trace information or to adjust the sampling strategies.

Procedure

By configuring the `spec.config.otel` field in the YAML files of ALB, FT, and Rule, you can add OTel-related configuration.

Additional Notes

Sampling Strategies

Parameter	Explanation
always on	Always report all tracing data.
always off	Never report tracing data.
traceid-ratio	Decide whether to report based on `traceid`. The format of `traceparent` is `xx-traceid-xx-flag`, where the first 16 characters of `traceid` represent a 32-bit hexadecimal integer. If this integer is less than `fraction` multiplied by 4294967295 (i.e., (2^32-1)), it will be reported.
parent-base	Decide whether to report based on the flag part of the traceparent in the request. When the flag is 01, it will be reported; for example: `curl -v "http://$ALB_IP/" -H 'traceparent: 00-xx-xx-01'`; when the flag is 02, it will not be reported; for example: `curl -v "http://$ALB_IP/" -H 'traceparent: 00-xx-xx-02'`.

Attributes

- **Resource Attributes**

  These attributes are reported by default.

Parameter	Description
hostname	The hostname of the ALB Pod
service.name	The name of the ALB
service.namespace	The namespace where the ALB resides
service.type	Default is ALB
service.instance.id	The name of the ALB Pod

- **Span Attributes**

  - Attributes reported by default:

Parameter	Description
http.status_code	Status code
http.request.resend_count	Retry count
alb.rule.rule_name	The name of the rule matched by this request
alb.rule.source_type	The type of the rule matched by this request, currently only Ingress
alb.rule.source_name	The name of the Ingress
alb.rule.source_ns	The namespace where the Ingress resides

- Attributes reported by default but can be excluded by modifying the flag.hide_upstream_attrs field:

Parameter	Description
alb.upstream.svc_name	The name of the Service (internal route) to which traffic is forwarded
alb.upstream.svc_ns	The namespace where the Service (internal route) being forwarded resides
alb.upstream.peer	The IP address and port of the Pod being forwarded to

- Attributes not reported by default but can be reported by modifying the flag.report_http_request_header field:

Parameter	Description
`**http.request.header.<header>**`	Request Header

- Attributes not reported by default but can be reported by modifying the flag.report_http_response_header field:

Parameter	Description
`**http.response.header.<header>**`	Response Header

Configuration Example

The following YAML configuration deploys an ALB and uses Jaeger as the OTel server, with Hotrod-proxy as the demonstration backend. By configuring Ingress rules, when clients request the ALB, the traffic will be forwarded to HotROD. Additionally, the communication between internal microservices of HotROD is also routed through the ALB.

1. Save the following YAML as a file named all.yaml.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hotrod
spec:
  replicas: 1
  selector:
    matchLabels:
      service.cpaas.io/name: hotrod
      service_name: hotrod
  template:
    metadata:
      labels:
        service.cpaas.io/name: hotrod
        service_name: hotrod
    spec:
      containers:
        - name: hotrod
          env:
            - name: PROXY_PORT
              value: '80'
            - name: PROXY_ADDR
              value: 'otel-alb.default.svc.cluster.local:'
            - name: OTEL_EXPORTER_OTLP_ENDPOINT
              value: 'http://jaeger.default.svc.cluster.local:4318'
          image: theseedoaa/hotrod-with-proxy:latest
          imagePullPolicy: IfNotPresent
          command: ['/bin/hotrod', 'all', '-v']
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: hotrod-frontend
spec:
  ingressClassName: otel-alb
  rules:
    - http:
        paths:
          - backend:
              service:
                name: hotrod
                port:
                  number: 8080
```

```
            path: /dispatch
            pathType: ImplementationSpecific
          - backend:
              service:
                name: hotrod
                port:
                  number: 8080
            path: /frontend
            pathType: ImplementationSpecific
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: hotrod-customer
spec:
  ingressClassName: otel-alb
  rules:
    - http:
        paths:
          - backend:
              service:
                name: hotrod
                port:
                  number: 8081
            path: /customer
            pathType: ImplementationSpecific
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: hotrod-route
spec:
  ingressClassName: otel-alb
  rules:
    - http:
        paths:
          - backend:
              service:
                name: hotrod
                port:
                  number: 8083
            path: /route
            pathType: ImplementationSpecific
---
```

```
apiVersion: v1
kind: Service
metadata:
  name: hotrod
spec:
  internalTrafficPolicy: Cluster
  ipFamilies:
    - IPv4
  ipFamilyPolicy: SingleStack
  ports:
    - name: frontend
      port: 8080
      protocol: TCP
      targetPort: 8080
    - name: customer
      port: 8081
      protocol: TCP
      targetPort: 8081
    - name: router
      port: 8083
      protocol: TCP
      targetPort: 8083
  selector:
    service_name: hotrod
  sessionAffinity: None
  type: ClusterIP
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: jaeger
spec:
  replicas: 1
  selector:
    matchLabels:
      service.cpaas.io/name: jaeger
      service_name: jaeger
  template:
    metadata:
      labels:
        service.cpaas.io/name: jaeger
        service_name: jaeger
    spec:
      containers:
```

```
        - name: jaeger
          env:
            - name: LOG_LEVEL
              value: debug
          image: jaegertracing/all-in-one:1.58.1
          imagePullPolicy: IfNotPresent
      hostNetwork: true
      tolerations:
        - operator: Exists
---
apiVersion: v1
kind: Service
metadata:
  name: jaeger
spec:
  internalTrafficPolicy: Cluster
  ipFamilies:
    - IPv4
  ipFamilyPolicy: SingleStack
  ports:
    - name: http
      port: 4318
      protocol: TCP
      targetPort: 4318
  selector:
    service_name: jaeger
  sessionAffinity: None
  type: ClusterIP
---
apiVersion: crd.alauda.io/v2
kind: ALB2
metadata:
  name: otel-alb
spec:
  config:
    loadbalancerName: otel-alb
    otel:
      enable: true
      exporter:
        collector:
          address: 'http://jaeger.default.svc.cluster.local:4318'
          request_timeout: 1000
    projects:
      - ALL_ALL
```

```
    replicas: 1
    resources:
      alb:
        limits:
          cpu: 200m
          memory: 2Gi
        requests:
          cpu: 50m
          memory: 128Mi
      limits:
        cpu: '1'
        memory: 1Gi
      requests:
        cpu: 50m
        memory: 128Mi
  type: nginx
```

2. In the CLI tool, execute the following command to deploy Jaeger, ALB, HotROD, and all necessary CRs for testing.

```
kubectl apply ./all.yaml
```

3. Execute the following command to get the access address of Jaeger.

```
export JAEGER_IP=$(kubectl get po -A -o wide |grep jaeger | awk '{print $7}');echo "http://$JAEGER_IP:16686"
```

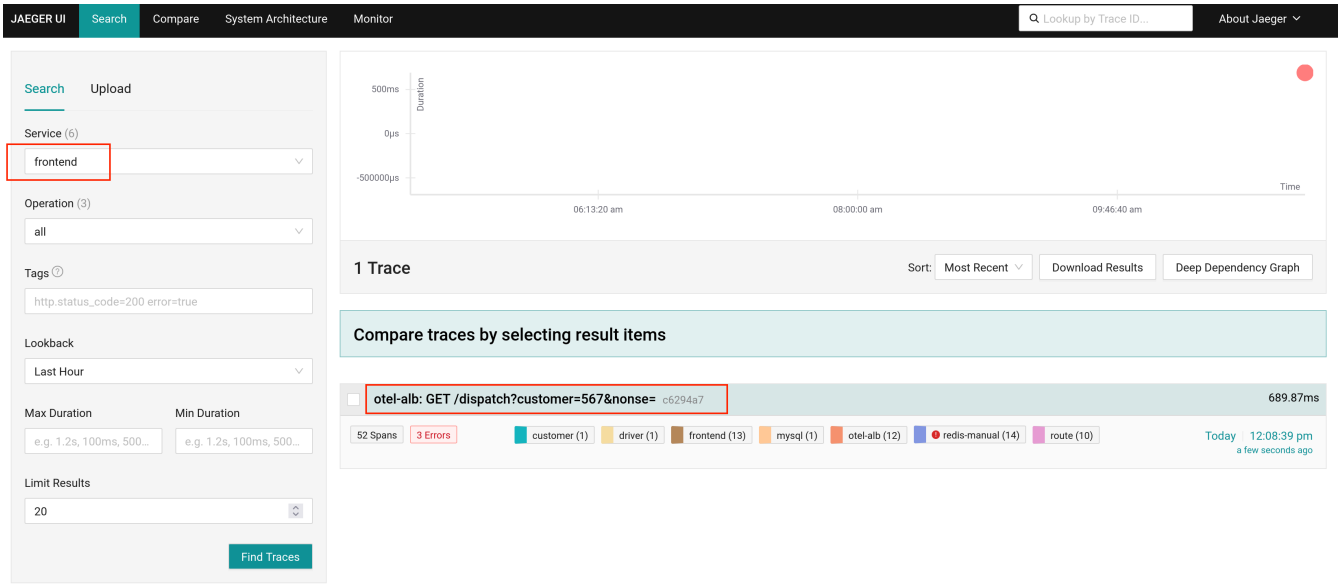
4. Execute the following command to obtain the access address of otel-alb.

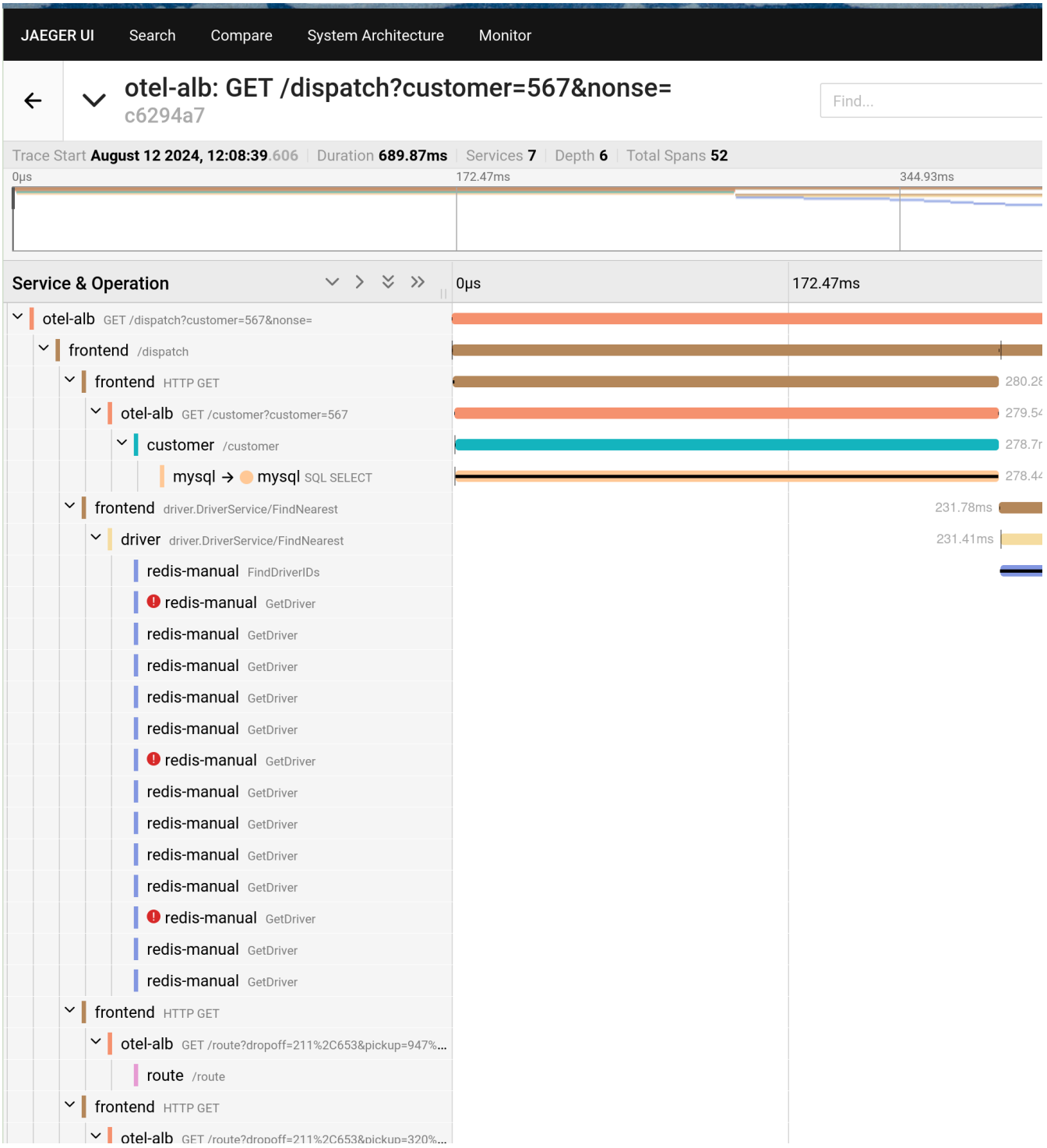
```
export ALB_IP=$(kubectl get po -A -o wide|grep otel-alb | awk '{print $7}');echo $ALB_IP
```

5. Execute the following command to send a request to HotROD via ALB. Here, ALB will report the Trace to Jaeger.

```
curl -v "http://<$ALB_IP>:80/dispatch?customer=567&nonse=" # Replace <$ALB_IP> in the command with the access address of otel-alb obtained in the previous procedure
```

6. Open the access address of Jaeger obtained in Step 3 to view the results.





Trademark Notice

NGINX® is a registered trademark of F5, Inc. in the U.S. and in other countries.

Alauda is an independent vendor. This product is not affiliated with, endorsed by, or sponsored by F5, Inc. All trademarks are the property of their respective owners and are used here for identification purposes only.

TCP/HTTP Keepalive

TOC

Basic Concepts

CRD

Trademark Notice

Basic Concepts

1. ALB supports keepalive configuration at the port level. It can be configured on the frontend.
2. Keepalive is **between the client and ALB, not between ALB and the backend**.
3. It is implemented through the Nginx configuration, and Nginx **needs and will automatically reload** when the configuration is changed.
4. TCP keepalive and HTTP keepalive are two different concepts:
    1. **TCP keepalive** is a TCP protocol feature that sends periodic probe packets to check if the connection is still alive when there is no data transmission. It helps detect and clean up dead connections.
    2. **HTTP keepalive** (also known as persistent connections) allows multiple HTTP requests to reuse the same TCP connection, avoiding the overhead of establishing new connections. This improves performance by reducing latency and resource usage.

CRD

```
keepalive:
  properties:
    http:
      description: Downstream L7 keepalive
      properties:
        header_timeout:
          description: Keepalive header timeout. Default is not set.
          type: string
        requests:
          description: Keepalive requests. Default is 1000.
          type: integer
        timeout:
          description: Keepalive timeout. Default is 75s.
          type: string
      type: object
    tcp:
      description: TCPKeepAlive defines TCP keepalive parameters (SO_KEEP
ALIVE)
      properties:
        count:
          description: The TCP_KEEPCNT socket option.
          type: integer
        idle:
          description: The TCP_KEEPIDLE socket option.
          type: string
        interval:
          description: The TCP_KEEPINTVL socket option.
          type: string
      type: object
  type: object
```

It can only be configured on the Frontend `.spec.config.keepalive`.

Trademark Notice

NGINX® is a registered trademark of F5, Inc. in the U.S. and in other countries.

Alauda is an independent vendor. This product is not affiliated with, endorsed by, or sponsored by F5, Inc. All trademarks are the property of their respective owners and are used here for identification purposes only.

ALB with Ingress-NGINX Annotation Compatibility

WARNING

ALB has been deprecated. Please use the ingress-nginx-operator if you need to use ingress-nginx annotations.

Basic concepts

ingress-nginx is a commonly used Ingress Controller in Kubernetes, and defines many annotations to implement various functions beyond the official ingress definition.

ALB supports partial of those annotations.

Supported ingress-nginx annotations

Name	type	Support (v supports x does not support o partially supports or can be achieved by configuration)
nginx.ingress.kubernetes.io/app-root	string	x

Name	type	Support (v supports x does not support o partially supports or can be achieved by configuration)
nginx.ingress.kubernetes.io/affinity	cookie	o ingress does not support. alb rule can configure cookie hash
nginx.ingress.kubernetes.io/use-regex	bool	
nginx.ingress.kubernetes.io/affinity-mode	"balanced" or "persistent"	o ingress does not support. alb rule can configure session persistence
nginx.ingress.kubernetes.io/affinity-canary-behavior	"sticky" or "legacy"	o ingress does not support. alb rule can configure session persistence
nginx.ingress.kubernetes.io/auth-realm	string	v auth
nginx.ingress.kubernetes.io/auth-secret	string	v auth
nginx.ingress.kubernetes.io/auth-secret-type	string	v auth
nginx.ingress.kubernetes.io/auth-type	"basic" or "digest"	v auth
nginx.ingress.kubernetes.io/auth-tls-secret	string	x
nginx.ingress.kubernetes.io/auth-tls-verify-depth	number	x
nginx.ingress.kubernetes.io/auth-tls-verify-client	string	x

Name	type	Support (v supports x does not support o partially supports or can be achieved by configuration)
nginx.ingress.kubernetes.io/auth-tls-error-page	string	x
nginx.ingress.kubernetes.io/auth-tls-pass-certificate-to-upstream	"true" or "false"	x
nginx.ingress.kubernetes.io/auth-tls-match-cn	string	x
nginx.ingress.kubernetes.io/auth-url	string	v
nginx.ingress.kubernetes.io/auth-cache-key	string	x
nginx.ingress.kubernetes.io/auth-cache-duration	string	x
nginx.ingress.kubernetes.io/auth-keepalive	number	x
nginx.ingress.kubernetes.io/auth-keepalive-share-vars	"true" or "false"	x
nginx.ingress.kubernetes.io/auth-keepalive-requests	number	x
nginx.ingress.kubernetes.io/auth-keepalive-timeout	number	x
nginx.ingress.kubernetes.io/auth-proxy-set-headers	string	v
nginx.ingress.kubernetes.io/auth-snippet	string	x
nginx.ingress.kubernetes.io/enable-global-auth	"true" or "false"	o auth

Name	type	Support (v supports x does not support o partially supports or can be achieved by configuration)
nginx.ingress.kubernetes.io/backend-protocol	string	v
nginx.ingress.kubernetes.io/canary	"true" or "false"	x
nginx.ingress.kubernetes.io/canary-by-header	string	x
nginx.ingress.kubernetes.io/canary-by-header-value	string	x
nginx.ingress.kubernetes.io/canary-by-header-pattern	string	x
nginx.ingress.kubernetes.io/canary-by-cookie	string	x
nginx.ingress.kubernetes.io/canary-weight	number	x
nginx.ingress.kubernetes.io/canary-weight-total	number	x
nginx.ingress.kubernetes.io/client-body-buffer-size	string	x
nginx.ingress.kubernetes.io/configuration-snippet	string	x
nginx.ingress.kubernetes.io/custom-http-errors	[]int	x
nginx.ingress.kubernetes.io/custom-headers	string	o
nginx.ingress.kubernetes.io/default-backend	string	o can use ingress's default-backend

Name	type	Support (v supports x does not support o partially supports or can be achieved by configuration)
nginx.ingress.kubernetes.io/enable-cors	"true" or "false"	v
nginx.ingress.kubernetes.io/cors-allow-origin	string	v
nginx.ingress.kubernetes.io/cors-allow-methods	string	v
nginx.ingress.kubernetes.io/cors-allow-headers	string	v
nginx.ingress.kubernetes.io/cors-expose-headers	string	x
nginx.ingress.kubernetes.io/cors-allow-credentials	"true" or "false"	x
nginx.ingress.kubernetes.io/cors-max-age	number	x
nginx.ingress.kubernetes.io/force-ssl-redirect	"true" or "false"	v redirect
nginx.ingress.kubernetes.io/from-to-www-redirect	"true" or "false"	x
nginx.ingress.kubernetes.io/http2-push-preload	"true" or "false"	x
nginx.ingress.kubernetes.io/limit-connections	number	x
nginx.ingress.kubernetes.io/limit-rps	number	x
nginx.ingress.kubernetes.io/global-rate-limit	number	x

Name	type	Support (v supports x does not support o partially supports or can be achieved by configuration)
nginx.ingress.kubernetes.io/global-rate-limit-window	duration	x
nginx.ingress.kubernetes.io/global-rate-limit-key	string	x
nginx.ingress.kubernetes.io/global-rate-limit-ignored-cidrs	string	x
nginx.ingress.kubernetes.io/permanent-redirect	string	v redirect
nginx.ingress.kubernetes.io/permanent-redirect-code	number	v redirect
nginx.ingress.kubernetes.io/temporal-redirect	string	v redirect
nginx.ingress.kubernetes.io/preserve-trailing-slash	"true" or "false"	x
nginx.ingress.kubernetes.io/proxy-body-size	string	x
nginx.ingress.kubernetes.io/proxy-cookie-domain	string	x
nginx.ingress.kubernetes.io/proxy-cookie-path	string	x
nginx.ingress.kubernetes.io/proxy-connect-timeout	number	v timeout
nginx.ingress.kubernetes.io/proxy-send-timeout	number	v timeout

Name	type	Support (v supports x does not support o partially supports or can be achieved by configuration)
nginx.ingress.kubernetes.io/proxy-read-timeout	number	v timeout
nginx.ingress.kubernetes.io/proxy-next-upstream	string	x
nginx.ingress.kubernetes.io/proxy-next-upstream-timeout	number	x
nginx.ingress.kubernetes.io/proxy-next-upstream-tries	number	x
nginx.ingress.kubernetes.io/proxy-request-buffering	string	x
nginx.ingress.kubernetes.io/proxy-redirect-from	string	x
nginx.ingress.kubernetes.io/proxy-redirect-to	string	x
nginx.ingress.kubernetes.io/proxy-http-version	"1.0" or "1.1"	x
nginx.ingress.kubernetes.io/proxy-ssl-secret	string	x
nginx.ingress.kubernetes.io/proxy-ssl-ciphers	string	x
nginx.ingress.kubernetes.io/proxy-ssl-name	string	x
nginx.ingress.kubernetes.io/proxy-ssl-protocols	string	x
nginx.ingress.kubernetes.io/proxy-ssl-verify	string	x

Name	type	Support (v supports x does not support o partially supports or can be achieved by configuration)
nginx.ingress.kubernetes.io/proxy-ssl-verify-depth	number	x
nginx.ingress.kubernetes.io/proxy-ssl-server-name	string	x
nginx.ingress.kubernetes.io/enable-rewrite-log	"true" or "false"	x
nginx.ingress.kubernetes.io/rewrite-target	URI	v
nginx.ingress.kubernetes.io/satisfy	string	x
nginx.ingress.kubernetes.io/server-alias	string	x
nginx.ingress.kubernetes.io/server-snippet	string	x
nginx.ingress.kubernetes.io/service-upstream	"true" or "false"	x
nginx.ingress.kubernetes.io/session-cookie-change-on-failure	"true" or "false"	x
nginx.ingress.kubernetes.io/session-cookie-conditional-samesite-none	"true" or "false"	x
nginx.ingress.kubernetes.io/session-cookie-domain	string	x
nginx.ingress.kubernetes.io/session-cookie-expires	string	x
nginx.ingress.kubernetes.io/session-cookie-max-age	string	x

Name	type	Support (v supports x does not support o partially supports or can be achieved by configuration)
nginx.ingress.kubernetes.io/session-cookie-name	string	x
nginx.ingress.kubernetes.io/session-cookie-path	string	x
nginx.ingress.kubernetes.io/session-cookie-samesite	string	x
nginx.ingress.kubernetes.io/session-cookie-secure	string	x
nginx.ingress.kubernetes.io/ssl-redirect	"true" or "false"	v
nginx.ingress.kubernetes.io/ssl-passthrough	"true" or "false"	x
nginx.ingress.kubernetes.io/stream-snippet	string	x
nginx.ingress.kubernetes.io/upstream-hash-by	string	x
nginx.ingress.kubernetes.io/x-forwarded-prefix	string	x
nginx.ingress.kubernetes.io/load-balance	string	x
nginx.ingress.kubernetes.io/upstream-vhost	string	v
nginx.ingress.kubernetes.io/denylist-source-range	CIDR	o can achieve similar effect through modsecurity
nginx.ingress.kubernetes.io/whitelist-source-range	CIDR	o can achieve similar effect through modsecurity

Name	type	Support (v supports x does not support o partially supports or can be achieved by configuration)
nginx.ingress.kubernetes.io/proxy-buffering	string	x
nginx.ingress.kubernetes.io/proxy-buffers-number	number	x
nginx.ingress.kubernetes.io/proxy-buffer-size	string	x
nginx.ingress.kubernetes.io/proxy-max-temp-file-size	string	x
nginx.ingress.kubernetes.io/ssl-ciphers	string	x
nginx.ingress.kubernetes.io/ssl-prefer-server-ciphers	"true" or "false"	x
nginx.ingress.kubernetes.io/connection-proxy-header	string	x
nginx.ingress.kubernetes.io/enable-access-log	"true" or "false"	o default enable access_log, format is fixed
nginx.ingress.kubernetes.io/enable-opentelemetry	"true" or "false"	v otel
nginx.ingress.kubernetes.io/opentelemetry-trust-incoming-span	"true" or "false"	v otel
nginx.ingress.kubernetes.io/enable-modsecurity	bool	v modsecurity
nginx.ingress.kubernetes.io/enable-owasp-core-rules	bool	v modsecurity

Name	type	Support (v supports x does not support o partially supports or can be achieved by configuration)
nginx.ingress.kubernetes.io/modsecurity-transaction-id	string	v modsecurity
nginx.ingress.kubernetes.io/modsecurity-snippet	string	v modsecurity
nginx.ingress.kubernetes.io/mirror-request-body	string	x
nginx.ingress.kubernetes.io/mirror-target	string	x
nginx.ingress.kubernetes.io/mirror-host	string	x

Trademark Notice

NGINX® is a registered trademark of F5, Inc. in the U.S. and in other countries.

Alauda is an independent vendor. This product is not affiliated with, endorsed by, or sponsored by F5, Inc. All trademarks are the property of their respective owners and are used here for identification purposes only.

ALB Monitoring

TOC

Terminology
Procedure
Monitoring Metrics
ALB Traffic Monitoring
ALB Resource Usage
Ingress, HTTPRoute, Rule Traffic Monitoring

Terminology

Term	Description
ALB	A self-developed layer-7 load balancer by the platform.

Procedure

1. Go to **Administrator**.
2. In the left navigation bar, click on **Operation Center** > **Monitoring** > **Monitoring Dashboard**.

3. Click on **Cluster** at the top of the page to switch to the cluster you want to monitor.
4. Click on **Switch** in the upper right corner of the page.
5. You can enter the **ALB Status** monitoring dashboard through the following two methods:
   - Method 1: Click on the **container-platform** card to expand the monitoring directory, then click on the **ALB Status** name to enter the monitoring dashboard. You can set this monitoring dashboard as the main dashboard if needed.
   - Method 2: Enter a keyword (e.g., alb) in the search box and search, then click on the **ALB Status** name to enter the monitoring dashboard. You can set this monitoring dashboard as the main dashboard if needed.
6. View various monitoring metrics through the dashboard.
   - **Select the namespace to monitor**: Click on the **namespace** at the top of the page to select the namespace to monitor, defaulting to all, meaning monitoring all namespaces.
   - **Select the ALB to monitor**: Click on the **name** at the top of the page to select the ALB to monitor, defaulting to all, meaning monitoring all ALBs.

Monitoring Metrics

Displays the monitoring metrics of total traffic, resource usage, Ingress (inbound rules), HTTPRoute (routing rules of type HTTPRoute), and Rule (rules that are neither Ingress nor HTTPRoute) for the selected ALB within the **last 5 minutes**.

Note: All data are monitoring data collected in the **last 5 minutes**.

ALB Traffic Monitoring

Monitoring Metric	Description
Active Connections	The number of active connections on the selected ALB.
Requests Per Second	The total number of requests received per second on the selected ALB.

Monitoring Metric	Description
Error Rate	The proportion of 4XX (such as 404) and 5XX error requests occurring per second on the selected ALB.
Latency	The average latency of requests on the selected ALB.

ALB Resource Usage

Monitoring Metric	Description
CPU Usage	The CPU usage of the selected ALB.
Memory Usage	The memory usage of the selected ALB.
Network Receive/Transmit	The network I/O throughput of the selected ALB.
Disk Read/Write Rate	The disk I/O throughput of the selected ALB.

Ingress, HTTPRoute, Rule Traffic Monitoring

Monitoring Metric	Description
QPS (Queries Per Second)	The number of requests received per second by the Ingress/HTTPRoute/Rule on the selected ALB, with the default unit being req/s.
Request BPS (Bytes Per Second)	The total size of requests received per second by the Ingress/HTTPRoute/Rule on the selected ALB.
Response BPS (Bytes Per Second)	The total size of responses sent by the Ingress/HTTPRoute/Rule on the selected ALB.
Error Rate	The percentage of errors that occurred when processing requests by the Ingress/HTTPRoute/Rule on the selected ALB.
P50, P90, P99	The response times for requests on the selected ALB, specifically the median response time. It indicates that 50%, 90%, and 99% of requests

Monitoring Metric	Description
	have a response time less than or equal to this value. **Note**: The principle of P50, P90, and P99 is to sort the collected data from smallest to largest and take the data values at the 50%, 90%, and 99% positions; thus, 50%, 90%, and 99% of the data collected are below this value. Percentiles help analyze the distribution of the data and identify various extreme situations.
Upstream P50, Upstream P90, Upstream P99	The request response times for upstream services. It indicates that 50%, 90%, and 99% of requests sent to upstream services have response times less than or equal to this value.

Network Security

Understanding Network Policy

Introduction

Evaluation of network policy by KUBE-OV

What is the difference between the Admin namespace scope?

Admin Network Policy

Notes

Creating AdminNetworkPolicy or Baseline

Creating AdminNetworkPolicy or Baseline

Network Po

Creating Netwo

Creating Netwo

Reference

Understanding Network Policy APIs

TOC

Introduction

Evaluation of network policy by KUBE-OVN

What is the difference between the AdminNetworkPolicy with scope and the NetworkPolicy with namespace scope?

Introduction

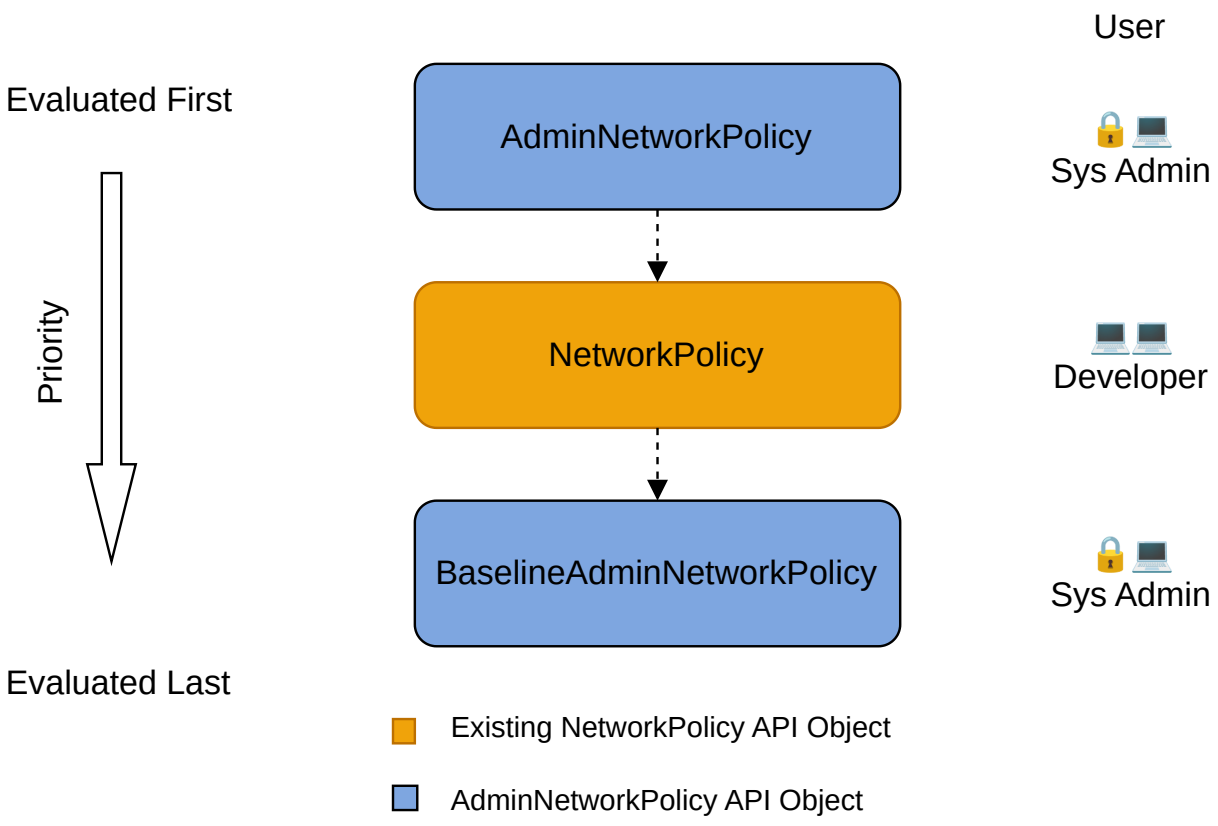
Network security in Kubernetes-style clusters involves controlling which workloads (Pods, namespaces) can communicate and how. At its core are network-policy APIs that enable segmentation (layer 3/4) and traffic control. Over time more advanced API layers have emerged to support cluster-wide guardrails, multi-tenancy enforcement, zero-trust models and application-layer (layer 7) filtering.

This article focuses on the API model v1alpha1: AdminNetworkPolicy (ANP) and BaselineAdminNetworkPolicy (BANP), their place among network policy APIs, how they relate to the standard NetworkPolicy, and how you might use them today while planning for future versions.

Evaluation of network policy by KUBE-OVN

When multiple network policies are applied, they follow a strict priority order: Admin Network Policy takes precedence over Network Policy, which in turn takes precedence over Baseline Admin Network Policy.

The procedure is as follows:



What is the difference between the AdminNetworkPolicy with scope and the NetworkPolicy with namespace scope?

Dimension	Namespace-Scoped NetworkPolicy	Cluster-Scoped AdminNetworkPolicy
Scope	Applies only to a single namespace	Applies across the entire cluster
Audience / Ownership	Application developer or namespace owners	Platform administrators or cluster administrator

Dimension	Namespace-Scoped NetworkPolicy	Cluster-Scoped AdminNetworkPolicy
Primary Use	Workload-level microsegmentation (L3/L4)	Enforcing global or cross-namespace access rules
Control Target	Pods within a namespace selected by `podSelector`	Namespaces or Pods selected by top-level `subject`
Policy Priority	Medium priority — evaluated after admin-level rules	Higher priority — overrides namespace policies
Typical Use Cases	Allow frontend → backend, restrict DB access, etc.	Restrict inter-tenant traffic, enforce egress restrictions, set global protections
Visibility	Only affects the namespace where it is created	Can impact multiple namespaces simultaneously
Cross-Namespace Rules	Allowed only indirectly (via namespaceSelector)	First-class cross-namespace traffic control
Risk Surface	Misconfiguration affects one namespace	Misconfiguration may affect entire cluster traffic

Admin Network Policy

INFO

The platform now provides two different UIs for Cluster Network Policies. The old one is maintained for compatibility reasons, while the new one is more flexible and provides a native YAML editor. We recommend using the new version.

Please contact the platform administrator to enable the `cluster-network-policy` and `cluster-network-policy-next` feature-gate to access the new UI.

TOC

Notes

Creating AdminNetworkPolicy or BaselineAdminNetworkPolicy by using the web console

Creating AdminNetworkPolicy or BaselineAdminNetworkPolicy by using the CLI

Notes

- Currently only Kube-OVN CNI supports admin network policies.
- In Kube-OVN network mode, this feature is a Technology Preview. The upstream API is still `policy.networking.k8s.io/v1alpha1` and continues to evolve, so evaluate this feature within its documented limits instead of treating it as production-supported. For the maturity definition, see Version and Lifecycle.

- Only one Baseline Admin Network Policy can exist in the cluster.

AdminNetworkPolicy

```yaml
# example-anp.yaml
apiVersion: policy.networking.k8s.io/v1alpha1
kind: AdminNetworkPolicy
metadata:
  name: example-anp
spec:
  priority: 3 (1)
  subject: (2)
    pods:
      namespaceSelector:
        matchLabels: {}
      podSelector:
        matchLabels:
          pod-template-hash: 55f66dd67d
  ingress:
    - name: ingress1
      action: Allow (3)
      ports:
        - portNumber:
            protocol: TCP
            port: 8090
      from: (4)
        - pods:
            namespaceSelector:
              matchLabels: {}
            podSelector:
              matchLabels:
                pod-template-hash: 55c84b59bb
  egress:
    - name: egress1
      action: Allow
      ports:
        - portNumber:
            protocol: TCP
            port: 8080
      to: (5)
        - networks:
            - 10.1.1.1/23
```

1. The lower the number, the higher the priority.
2. `subject`: At most one of namespace selector or pod selector can be specified.
3. `action`: The available values are Allow, Deny, and Pass. Allow for allowing traffic access, Deny for denying traffic access, Pass for allowing the traffic and skip subsequent low priority cluster network policies and continue to have the traffic handled by other policies (NetworkPolicy and BaselineAdminNetworkpolicy).
4. The available values are Namespace Selector, Pod Selector.
5. The available values are Namespace Selector, Pod Selector, Node Selector, IP Block.

BaselineAdminNetworkpolicy:

```yaml
# default.yaml
apiVersion: policy.networking.k8s.io/v1alpha1
kind: BaselineAdminNetworkPolicy
metadata:
  name: default (1)
spec:
  subject:
    pods:
      namespaceSelector:
        matchLabels: {}
      podSelector:
        matchLabels:
          pod-template-hash: 55c84b59bb
  ingress:
    - name: ingress1
      action: Allow
      ports:
        - portNumber:
            protocol: TCP
            port: 8888
      from:
        - pods:
            namespaceSelector:
              matchLabels: {}
            podSelector:
              matchLabels:
                pod-template-hash: 55f66dd67d
  egress:
    - name: egress1
      action: Allow (2)
      ports:
        - portNumber:
            protocol: TCP
            port: 8080
      to:
        - networks:
            - 3.3.3.3/23
```

(1) Only one baseline admin network policy with metadata.name=`default` can be created in the cluster.

(2) The available values are Allow, Deny.

Creating AdminNetworkPolicy or BaselineAdminNetworkPolicy by using the web console

1. Go to **Administrator**.
2. In the left navigation bar, click **Networking** > **Cluster Network Policies**.
3. Click **Create Admin Network Policies** or **Configure the Baseline Admin Network Policy**.
4. Follow the instructions below to complete the relevant configuration.

Area	Parameter	Description
Basic Information	Name	The name of the Admin Network Policy or Baseline Admin Network Policy.
	Priority	Determines the order in which policies are evaluated and applied. Lower numerical values indicate higher priority. **Note:** The baseline admin network policy does not have a priority.
Target Pod	Namespace Selector	Enter the labels of the target Namespaces in key-value form. If not set, the policy will apply to all Namespaces in the current cluster. When specified, the policy will only apply to pods within the namespaces that match these selectors.

Area	Parameter		Description
	Preview of Target Pods Affected by Current Policy		Click **Preview** to see the target Pods affected by this network policy.
	Pod Selector		Enter the labels of the target Pods in key-value form. If not set, the policy will apply to all Pods in the current namespace.
	Preview of Target Pods Affected by Current Policy		Click **Preview** to see the target Pods affected by this network policy.
Ingress	Traffic Action		Specifies how to handle incoming traffic to target Pods. Has three modes: **Allow** (permits traffic), **Deny** (blocks traffic), and **Pass** (skips all lower-priority admin network policies, allowing the traffic to be handled by Network Policy, or if no Network Policy exists, by Baseline Admin Network Policy). **Note:** The baseline admin network policy does not have action **Pass**.
	Rule **Description**: If multiple sources are added in the rule, there is a	Pod Selector	Matches namespaces or Pods with specified labels in the cluster; only matching Pods can access the target Pod. You can click **Preview** to see

Area	Parameter		Description
	logical **OR** relationship between them.		the Pods affected by the current rule. • If both namespace and Pod selectors are configured, their intersection will be taken, meaning Pods with specified labels will be selected from the specified namespaces. • If this item is not configured, all Pods in all namespaces in the cluster can access the target Pod by default.
		Namespace Selector	Matches Pods with specified labels in the current namespace; only matching Pods can access the target Pod. You can click **Preview** to see the Pods affected by the current rule. If this item is not configured, all Pods in the current namespace are allowed to access the target Pod by default.

Area	Parameter		Description
Egress	Ports		Matches traffic on specified protocols and ports; you can add numeric ports or port names on Pods. If this item is not configured, all ports will be matched.
	Rule **Description**: If multiple sources are added in the rule, there is a logical **OR** relationship between them.	Node Selector	Specifies which node IPs the target Pods are allowed to access. You can select nodes by their labels to control which node IPs are accessible from the Pods.
		IP Range	Specify CIDR ranges that target Pods are allowed to connect to. If this item is not configured, target Pods can connect to any IP by default.
		Other Parameters	Similar to the Ingress parameters, with the same configuration options and behavior.

Creating AdminNetworkPolicy or BaselineAdminNetworkPolicy by using the CLI

```
kubectl apply -f example-anp.yaml -f default.yaml
```

Network Policy

INFO

The platform now provides two different UIs for Network Policies. The old one is maintained for compatibility reasons, while the new one is more flexible and provides a native YAML editor. We recommend using the new version.

Please contact the platform administrator to enable the `network-policy-next` feature gate to access the new UI.

NetworkPolicy is a namespace-scoped Kubernetes resource and implemented by CNI plugins. Through network policies, you can control network traffic of Pods, achieving network isolation and reducing the risk of attacks.

By default, all Pods can communicate freely, allowing ingress and egress traffic from any source. When a NetworkPolicy is applied, the targeted Pods will only accept traffic that matches the spec.

WARNING

Network policies only apply to container traffic. They don't affect Pods running in **hostNetwork** mode.

Example NetworkPolicy:

```yaml
# example-network-policy.yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: example
  namespace: demo-1
  annotations:
    cpaas.io/display-name: test
spec:
  podSelector:
    matchLabels:
      pod-template-hash: 55c84b59bb
  ingress:
    - ports:
        - protocol: TCP
          port: 8989
      from: (1)
        - podSelector:
            matchLabels:
              kubevirt.io/vm: test
  egress:
    - ports:
        - protocol: TCP
          port: 80
      to:
        - ipBlock:
            cidr: 192.168.66.221/23
            except: []
  policyTypes:
    - Ingress
    - Egress
```

(1) `from` and 'to' peer support `namespaceSelector`, `podSelector`, 'ipBlock'

TOC

Creating NetworkPolicy by using the web console

Creating NetworkPolicy by using the CLI

Reference

Creating NetworkPolicy by using the web console

1. Enter **Container Platform**.
2. In the left navigation bar, click **Network** > **Network Policies**.
3. Click **Create Network Policy**.
4. Refer to the following instructions to complete the relevant configuration.

Area	Parameter	Description
Target Pod	Pod Selector	Enter the labels of the target Pods in key-value form; if not set, it will apply to all Pods in the current namespace.
	Preview of Target Pods Affected by Current Policy	Click **Preview** to see the target Pods affected by this network policy.
Ingress	Block all ingress traffic	Block all ingress traffic to the target Pod. **Note**: • If Ingress is added to the `spec.policyTypes` field in YAML without configuring specific rules, the **Block all ingress traffic** option will automatically be checked when switching back to the form. • If the `spec.ingress`, `spec.egress`, and `spec.policyTypes` fields are

Area	Parameter		Description
			simultaneously deleted in YAML, the **Block all ingress traffic** option will automatically be checked when switching back to the form.
	Rules **Description**: If multiple sources are added in the rules, there is a logical **OR** relationship between them.	Pods in Current Namespace	Match Pods with specified labels in the current namespace; only matched Pods can access the target Pod. You can click **Preview** to see the Pods affected by the current rule. If this item is not configured, all Pods in the current namespace are allowed to access the target Pod by default.
		Pods in Current Cluster	Match namespaces or Pods with specified labels in the cluster; only matched Pods can access the target Pod. You can click **Preview** to see the Pods affected by the current rule. • If both namespace and Pod selectors are configured, it will take the intersection of the two, meaning Pods with specified labels will be selected from the specified namespace. • If this item is not configured, all Pods from all namespaces in the cluster can access the target Pod by default.
		IP Range	Enter the CIDR that can access the target Pod and can exclude CIDR ranges that are not allowed

Area	Parameter	Description
		to access the target Pod. If this item is not configured, any traffic can access the target Pod. **Description**: You can add exclusion items in the form of *example_ip*/32 to exclude a single IP address.
	Port	Match traffic on specified protocols and ports; numeric ports or port names on Pods can be added. If this item is not configured, all ports will be matched.
Egress	Block all egress traffic	Block all egress traffic to the target Pod. **Note**: - If Egress is added to the `spec.policyTypes` field in YAML without configuring specific rules, the **Block all egress traffic** option will automatically be checked when switching back to the form.
	Other Parameters	Similar to the **Ingress** parameters, this will not be elaborated on here.

5. Click **Create**.

Creating NetworkPolicy by using the CLI

```
kubectl apply -f example-network-policy.yaml
```

Reference

If you want more details, check out the official docs on Network Policies↗.

Ingress and Load Balancing

Ingress and Load Balancing with Envoy Gateway

Overview

Network Flow Architecture

Configuring Routes

Configuring Ingress Cluster Traffic

Load Balancing Strategies

TLS and Security

Integration with MetalLB

Troubleshooting

Trademark Notice

Ingress and Load Balancing with Envoy Gateway

TOC

- Overview
- Network Flow Architecture
- Configuring Routes
  - Defining a GatewayClass
  - Creating a Gateway
  - Creating HTTP Routes
- Configuring Ingress Cluster Traffic
- Load Balancing Strategies
  - Default Round-Robin
  - Weighted Load Balancing
  - Session Affinity (Sticky Sessions)
- TLS and Security
  - TLS Termination
  - TLS Passthrough
  - mTLS Between Gateways and Services
- Integration with MetalLB
- Troubleshooting
- Trademark Notice

Overview

Envoy Gateway provides Layer 7 (L7) ingress and load balancing for Kubernetes clusters. It is based on the Kubernetes Gateway API and built atop Envoy Proxy, allowing advanced HTTP, HTTPS, gRPC, and TCP routing with full observability and policy control.

Envoy Gateway unifies the ingress traffic management model by defining GatewayClass, Gateway, and `Route` resources, offering a standard, portable alternative to legacy Ingress objects.

Network Flow Architecture

```
External Client (Browser / curl) https://www.example.com
   |
   ▼
DNS Resolution (https://www.example.com) → External IP (e.g. 34.23.88.11)
   |
   ▼
External Load Balancer [L4]
   |
   ▼
Envoy Gateway (Gateway API) [L7]
   |
   ▼
Kubernetes Service (ClusterIP) [L4]
   |
   ▼
Pod (Application)
```

- External LoadBalancer provides an external IP that maps traffic into the cluster.
- Envoy Gateway terminates TLS, applies routing rules, and forwards to backend services.
- Service (ClusterIP) distributes traffic to Pods across nodes.

This architecture replaces the “Ingress Controller” with a fully Gateway-API-compliant data plane.

Configuring Routes

Defining a GatewayClass

```yaml
apiVersion: gateway.networking.k8s.io/v1
kind: GatewayClass
metadata:
  name: envoy
spec:
  controllerName: gateway.envoyproxy.io/gatewayclass-controller
```

Creating a Gateway

```yaml
apiVersion: gateway.networking.k8s.io/v1
kind: Gateway
metadata:
  name: public-gateway
  namespace: gateway-system
spec:
  gatewayClassName: envoy
  listeners:
    - name: http
      protocol: HTTP
      port: 80
      allowedRoutes:
        namespaces:
          from: Same
    - name: https
      protocol: HTTPS
      port: 443
      tls:
        mode: Terminate
        certificateRefs:
          - kind: Secret
            name: site-cert
```

Learn more Configure Gateway API Gateway.

Creating HTTP Routes

```yaml
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: web-route
  namespace: default
spec:
  parentRefs:
    - name: public-gateway
      namespace: gateway-system
  hostnames:
    - 'app.example.com'
  rules:
    - matches:
        - path:
            type: PathPrefix
            value: /
      backendRefs:
        - name: web-service
          port: 8080
```

Learn more Configure Gateway API Route.

Explanation

- The Gateway exposes ports 80 and 443 for HTTP/S traffic.
- The HTTPRoute defines routing rules based on hostname and path.
- Multiple HTTPRoute objects can share the same Gateway.

Configuring Ingress Cluster Traffic

Envoy Gateway supports various external traffic entry modes, depending on your infrastructure.

Environment	Entry Type	Example
Cloud	`Service type=LoadBalancer`	Cloud provider assigns public IP
Bare Metal	MetalLB + `Service type=LoadBalancer`	MetalLB allocates external IP
Edge / Internal	NodePort or hostNetwork	For local or private access

Example Service exposing Envoy Gateway:

```yaml
apiVersion: v1
kind: Service
metadata:
  name: envoy-gateway
  namespace: gateway-system
spec:
  type: LoadBalancer
  selector:
    app: envoy
  ports:
    - name: http
      port: 80
      targetPort: 8080
    - name: https
      port: 443
      targetPort: 8443
```

You can verify the external IP assigned by running:

```bash
kubectl get svc envoy-gateway -n gateway-system
```

Load Balancing Strategies

Envoy Gateway provides flexible load-balancing methods through its backend references.

Default Round-Robin

```yaml
backendRefs:
  - name: app-v1
    port: 8080
  - name: app-v2
    port: 8080
```

Weighted Load Balancing

```yaml
backendRefs:
  - name: app-v1
    port: 8080
    weight: 80
  - name: app-v2
    port: 8080
    weight: 20
```

Session Affinity (Sticky Sessions)

Envoy supports cookie-based or source-IP-based session affinity via policy configuration:

```yaml
apiVersion: gateway.envoyproxy.io/v1alpha1
kind: BackendTrafficPolicy
metadata:
  name: web-policy
  namespace: default
spec:
  targetRef:
    group: gateway.networking.k8s.io
    kind: HTTPRoute
    name: web-route
  sessionAffinity:
    type: Cookie
    cookie:
      name: session_id
      ttl: 3600s
```

Learn more about BackendTrafficPolicy.

TLS and Security

TLS Termination

Handled at the Gateway listener level:

```
tls:
  mode: Terminate
  certificateRefs:
    - kind: Secret
      name: site-cert
```

TLS Passthrough

For end-to-end encryption where Envoy forwards encrypted traffic:

```
tls:
  mode: Passthrough
```

mTLS Between Gateways and Services

Advanced policy CRDs can enable mTLS or client certificate validation.

Integration with MetalLB

In bare-metal clusters without cloud load balancers:

1. Configure a MetalLB IPAddressPool and L2Advertisement.
2. Ensure the Envoy Gateway Service is of type LoadBalancer.
3. MetalLB assigns an external IP automatically.

4. DNS records can then map domains (e.g., app.example.com) to the assigned IP.

Troubleshooting

Issue	Possible Cause	Resolution
Gateway has no external IP	MetalLB misconfigured	Check `Service` type and MetalLB controller
HTTPRoute ignored	`parentRefs` mismatch	Ensure correct `Gateway` and namespace
TLS not working	Secret not found	Verify `certificateRefs` path
404 from Gateway	No matching host/path	Confirm `HTTPRoute` rules
Latency spikes	Overloaded proxy	Scale Envoy Gateway replicas

Trademark Notice

Envoy® is a registered trademark of The Linux Foundation.

MetalLB™ is a trademark of The Linux Foundation.

Alauda is an independent vendor. This product is not affiliated with, endorsed by, or sponsored by The Linux Foundation. All trademarks are the property of their respective owners and are used here for identification purposes only.

Guides

Configure Domain

Example Domain custom resource (CR)
Creating Domain by using the web console
Creating Domain by using the CLI
Use the Same Domain Across Multiple Clu
Subsequent Actions
Additional resources

Creating Certificates

Creating a certificate by using the web cor

Configure S

Why Service is
Example Cluste
Headless Servi

Configure Ingresses

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

Trademark Noti

Configure Subnets

IP Allocation Rules
Calico Network
Kube-OVN Network
Subnet Management

Configure Alauda Container Platform Load Ba

Prerequisites
Configure MetalLB Components
Configure an External IP Pool by using the web console
Configure BGP Peers by using the web console
Configure an External IP Pool with L2Advertisement or BGPA

Configure GatewayAPI Gateway

Overview
Prerequisites
Gateway Basics

Configure G

Overview

Create Gateway
View Gateway Details
Listener and Route Reference
Next Step
Trademark Notice

Prerequisites
Policy Attachme
Policy Attachme
Create Policies
SecurityPolicy

Configure GatewayAPI Route

Configure GatewayAPI ListenerSet

Overview
Prerequisites
Configuration
Attach a Route
Policy behavior
View ListenerSet Details
Next Step
Related Tasks
Trademark Notice

Configure E

Overview
Quota model
Create a quota

Configure ALB

ALB
Frontend

Configure Alauda Container Platform Node DNS Cache

Overview
Key Features
Important Notes
Installation
How It Works
Configuration
Known Risks and Temporary Workaround
Trademark Notice

Configure Alauda Container Platform Cluster

Overview
Configuration
Trademark Notice

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

- Example Domain custom resource (CR)
- Creating Domain by using the web console
- Creating Domain by using the CLI
- Use the Same Domain Across Multiple Clusters
  - Configure via Web Console
  - Configure via CLI
- Subsequent Actions
- Additional resources

Example Domain custom resource (CR)

```yaml
# test-domain.yaml
apiVersion: crd.alauda.io/v2
kind: Domain
metadata:
  name: '${random-unique-name}'
  annotations:
    cpaas.io/secret-ref: developer.test.cn-xfd8x (1)
  labels:
    cluster.cpaas.io/name: global
    project.cpaas.io/name: demo
spec:
  name: developer.test.cn
  kind: full
```

(1) 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	• Domain: A complete domain name, e.g., `developer.test.cn`. • Wildcard Domain: A wildcard domain with a wildcard (*) character, e.g., `*.test.cn`, which includes all subdomains under the domain `test.cn`.
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	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. **Notes**: • Binary file imports are not supported. • The bound certificate should meet the conditions of correct format, within the validity period, and signed for the domain name, etc. • After creating the bound certificate, the name format of the bound certificate is: domain name - random characters. • 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. • After creating the bound certificate, updating the certificate content is supported, but replacing other certificates is not supported.

5. Click **Create**.

Creating Domain by using the CLI

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

Use the Same Domain Across Multiple Clusters

You can configure the same domain to be used across multiple clusters by creating separate Domain resources in the global cluster with the identical `spec.name` value but different `cluster.cpaas.io/name` labels.

Configure via Web Console

1. Follow the steps in Creating Domain by using the web console.
2. Create two domain resources with the **same domain name** (e.g., `app.example.com`).
3. For each domain, select a different **Allocate Cluster** (e.g., cluster-a and cluster-b).

Configure via CLI

Create two Domain custom resources in the global cluster with identical `spec.name` but different `cluster.cpaas.io/name` labels:

> **NOTE**
>
> The domain name (`spec.name`) must be identical in both Domain resources, but the resource `metadata.name` should be unique. Both resources are created in the global cluster.

Domain for Cluster A:

```yaml
apiVersion: crd.alauda.io/v2
kind: Domain
metadata:
  name: '${random-unique-name}'
  labels:
    cluster.cpaas.io/name: cluster-a
    project.cpaas.io/name: project-a
spec:
  name: app.example.com  # Same domain name
  kind: full
```

Domain for Cluster B:

```yaml
apiVersion: crd.alauda.io/v2
kind: Domain
metadata:
  name: '${random-unique-name}'
  labels:
    cluster.cpaas.io/name: cluster-b
    project.cpaas.io/name: project-a
spec:
  name: app.example.com  # Same domain name as Cluster A
  kind: full
```

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
Assign Project	• All Projects: Assign the certificate for use in all projects associated with the current cluster. • Specified Project: Assign the certificate for use in the specified project. • 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 **Update Project** operation.
Public Key	This refers to tls.crt. When importing the public key, binary files are not supported.
Private Key	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 Servce
- 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

Trademark Notice

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:

```yaml
# 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
Virtual IP Address	If enabled, a ClusterIP will be allocated for this Service, which can be used for service discovery within the cluster. If disabled, a Headless Service will be created, which is usually used by **StatefulSet**.
Type	• **ClusterIP**: Exposes the Service on a cluster-internal IP. Choosing this value makes the Service only reachable from within the cluster. • **NodePort**: Exposes the Service on each Node's IP at a static port (the NodePort). • **ExternalName**: Maps the Service to the contents of the externalName field (for example, to the hostname api.foo.bar.example). • **LoadBalancer**: 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.
Target Component	• **Workload**: The Service will forward requests to a **specific** workload, which matches the labels like `project.cpaas.io/name: projectname` and `service.cpaas.io/name: deployment-name`. • **Virtualization**: The Service will forward requests to a **specific** virtual machine or virtual machine group. • **Label Selector**: The Service will forward requests to a **certain type** of workload with specified labels, for example, `environment: release`.
Port	Used to configure the port mapping for this Service. In the following example, other podss 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 *redis* of the target component's pods.

Parameter	Description
	• **Protocol**: The protocol used by the Service, supported protocols include: `TCP`, `UDP`, `HTTP`, `HTTP2`, `HTTPS`, `gRPC`. • **Service Port**: The service port number exposed by the Service within the cluster, that is, Port, e.g., *80*. • **Container Port**: The target port number (or name) that the service port maps to, that is, targetPort, e.g., *6379* or *redis*. • **Service Port Name**: Will be generated automatically. The format is `<protocol>-<service port>-<container port>`, for example: *tcp-80-6379* or *tcp-80-redis*.
Session Affinity	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

```yaml
# 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

```yaml
# 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.

```yaml
# 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)
AGE
nginx-service   LoadBalancer   10.0.2.57        34.122.45.100   80:3000
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 Servce

```yaml
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
service.beta.kubernetes.io/aws-load-balancer-type	external: Use the official AWS LoadBalancer Controller.	Specifies the controller for the LoadBalancer type. **Note**: Please contact the platform administrator in advance to deploy the AWS LoadBalancer Controller.
service.beta.kubernetes.io/aws-load-balancer-nlb-target-type	• instance: Traffic will be sent to the pods via NodePort. • ip: Traffic routes directly to the pods (the cluster must use Amazon VPC CNI).	Specifies how traffic reaches the pods.

Key	Value	Description
service.beta.kubernetes.io/aws-load-balancer-scheme	• internal: Private network. • internet-facing: Public network.	Specifies whether to use a private network or a public network.
service.beta.kubernetes.io/aws-load-balancer-ip-address-type	• IPv4 • dualstack	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	Example: `{"type":"public","bandwidth_name":"cce-bandwidth-1551163379627","bandwidth_chargemode":"bandwidth","bandwidt["cn-north-4b"],"l4_flavor_name":"L4_flavor.elb.s1.small"}` **Note**: Please read the Filling Instructions ↗ first and adjust the exampl
kubernetes.io/elb.subnet-id	

Key	Value
kubernetes.io/elb.class	• union: Shared load balancing. • performance: Exclusive load balancing, only supported in Kuberne
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	• true: Private network. • false: Public network.	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/l4-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 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`:

```yaml
# 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
   ORT(S)        AGE
   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`

Trademark Notice

NGINX® is a registered trademark of F5, Inc. in the U.S. and in other countries.

MetalLB™ is a trademark of The Linux Foundation.

Alauda is an independent vendor. This product is not affiliated with, endorsed by, or sponsored by The Linux Foundation or F5, Inc. All trademarks are the property of their respective owners and are used here for identification purposes only.

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:

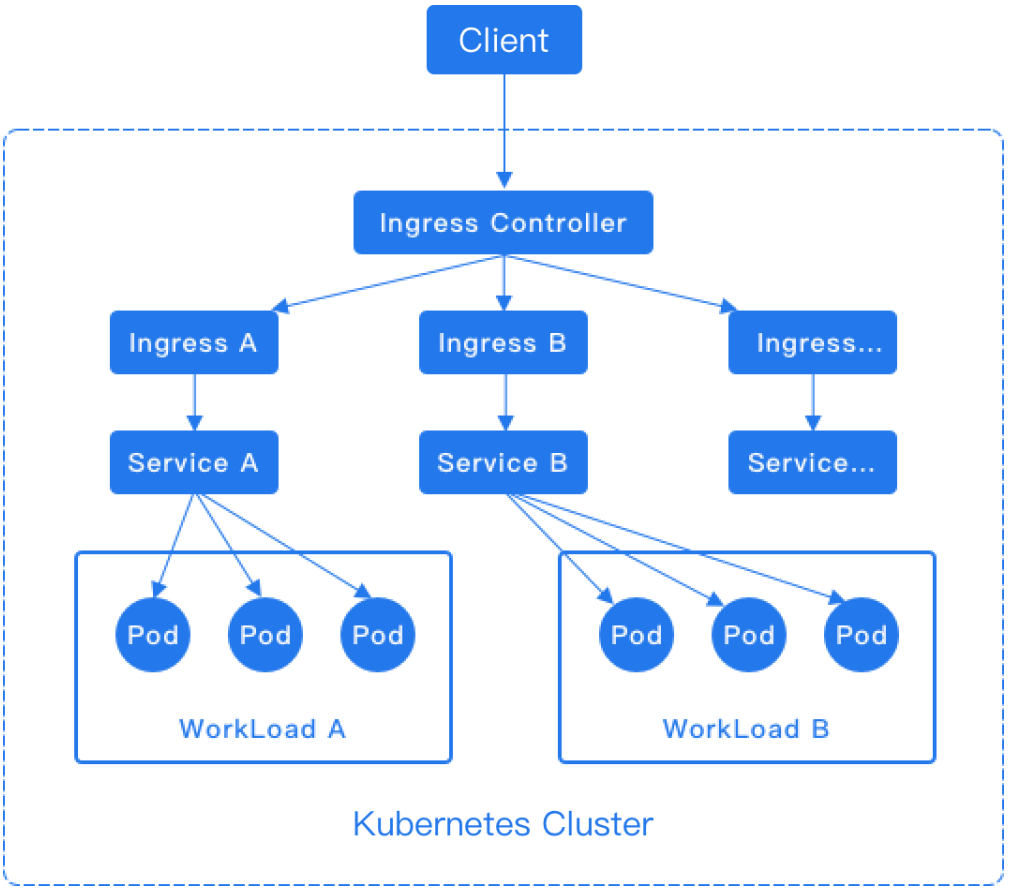
Creating a Ingress by using the web console

Creating a Ingress by using the CLI

Trademark Notice

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:

```yaml
# nginx-ingress.yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: nginx-ingress
  namespace: k-1
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: / 1
spec:
  ingressClassName: nginx 2
  rules:
    - host: demo.local 3
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: nginx-service
                port:
                  number: 80
```

1 To see more configurations please refer to nginx-configuration ↗.

2 `nginx` to using `ingress-nginx` controller, `$alb_name` to use alb as ingress controller.

3 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
Ingress Class	Ingresses can be implemented by different controllers with different `IngressClass` name. If multiple ingress controllers are available on the platform, the user can select which one to use with this option.
Domain Name	Hosts can be precise matches (for example `foo.bar.com`) or a wildcard (for example `*.foo.com`). The domain names available are allocated by platform administrator.
Certificates	TLS secret or Certificates allocated by platform administrator.
Match Type and **Path**	• **Prefix**: Matches path prefixes, e.g., `/abcd` can match `/abcd/efg` or `/abcde`. • **Exact**: Matches exact paths, e.g., `/abcd`. • **Implementation specific**: If you are using a custom Ingress controller to manage the Ingress rules, you may choose to have the controller decide.
Service	External traffic will be forwarded to this Service.
Service Port	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
```

Trademark Notice

NGINX® is a registered trademark of F5, Inc. in the U.S. and in other countries.

Alauda is an independent vendor. This product is not affiliated with, endorsed by, or sponsored by F5, Inc. All trademarks are the property of their respective owners and are used here for identification purposes only.

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

```yaml
# 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 **Networking** > **Container Network** > **Subnets**.
3. Click **Create Subnet**.
4. Refer to the following instructions to configure the relevant parameters.

Parameter	Description
CIDR	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. **Note**: For the correspondence between CIDR and BlockSize, please refer to Reference Content.
Encapsulation Protocol	Select the encapsulation protocol. **IPIP** is not supported in dual-stack mode. • **IPIP**: Implements inter-segment communication using the IPIP protocol. • **VXLAN (Alpha)**: Implements inter-segment communication using the VXLAN protocol. • **No Encapsulation**: Directly connected through routing forwarding.

Parameter	Description
Encapsulation Mode	When the encapsulation protocol is **IPIP** or **VXLAN**, the encapsulation mode must be set, defaulting to **Always**. • **Always**: Always enable IPIP / VXLAN tunnels. • **Cross Subnet**: 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.
Outbound Traffic NAT	Choose whether to enable outbound traffic NAT (Network Address Translation), which is enabled by default. It is primarily used to set the access addresses exposed to the external network when the subnet container group accesses the external network. 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.

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

CIDR	blockSize Size	Number of Hosts	Size of a Single IP Pool
`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

```yaml
# 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 **Networking** > **Container Network** > **Subnets**.
3. Click on **Create Subnet**.
4. Refer to the following instructions to configure the related parameters.

Parameter	Description
Network Segment	After assigning the subnet to the project or namespace, IPs within this segment will be randomly allocated for use by Pods.
Reserved IP	The set reserved IP will not be automatically allocated. For example, it can be used as the IP address for computing components' **fixed IP**.
Gateway Type	Select the type of gateway for the subnet to control the outbound traffic. - **Distributed**: Each host in the cluster can act as an outbound node for Pods on the current host, enabling distributed egress. - **Centralized**: All Pods in the cluster use one or more specific hosts as outbound nodes, facilitating external auditing and firewall control. Setting multiple centralized **gateway nodes** can achieve high availability.
ECMP (Alpha)	When choosing a **Centralized** 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. **Note**: Please enable ECMP-related features in advance.
Gateway Nodes	When using a **Centralized** gateway, select one or more specific hosts as gateway nodes.
Outbound Traffic NAT	Choose whether to enable outbound traffic NAT (Network Address Translation). By default, it is enabled. It is mainly used to set the access address exposed to the external network when the Pods in the subnet access the internet. When outbound traffic NAT is enabled, the host IP will be used as the

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 **Networking** > **Container Network** > **Bridge Networks**.
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
Default Network Card Name	By default, the target Pod will use this as the bridge network card for intercommunication with the physical network.
Configure Network Card by Node	The target Pods on the configured nodes will bridge to the specified network card instead of the default network card.
Exclude Nodes	When nodes are excluded, all Pods scheduled to these nodes will not bridge to any network card on these nodes. **Note**: 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.

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 **Networking** > **Container Network** > **VLANs**.
3. Click **Add VLAN**.
4. Configure the relevant parameters based on the following instructions.

Parameter	Description
VLAN ID	The unique identifier for this VLAN, which will be used to differentiate different virtual LANs.
Bridge Network	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

```yaml
# 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 **Networking** > **Container Network** > **Subnets**.
3. Click **Create Subnet**.
4. Configure the relevant parameters based on the following instructions.

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

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.

If workloads in an Underlay subnet require IPv4 multicast communication, see Configure IPv4 Multicast for Kube-OVN Underlay Networks.

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 **Networking** > **Container Network** > **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 **Networking** > **Container Network** > **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 **Networking** > **Container Network** > **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 **Networking** > **Container Network** > **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 **Networking** > **Container Network** > **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"
  }
]'
```

NOTE

Expanding the default Pod subnet

If a cluster is running out of Pod IP addresses, you can expand the default Pod subnet (`ovn-default`) in the same way. Existing Pods keep their addresses, and new Pods receive addresses from the enlarged range.

A subnet can only be enlarged while keeping the same network base address (the network prefix is made shorter); you cannot change the base address or shrink the range. If the default Pod subnet cannot be enlarged in place—for example, a larger range would not align to the current base address—add an additional subnet and bind it to the namespaces that need more addresses instead.

The Service CIDR cannot be changed after the cluster is created. Plan the Service network size before installation.

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 **Networking** > **Container Network** > **Subnets**.
3. Click ⋮ > **Delete**, and proceed with the deletion.

Delete Subnet by using the CLI

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

Configure Alauda Container Platform Load Balancer

TOC

Prerequisites
Configure MetalLB Components
Configure an External IP Pool by using the web console
Configure BGP Peers by using the web console
Configure an External IP Pool with L2Advertisement or BGPAdvertisement by using the CLI
Monitor MetalLB
View the built-in MetalLB dashboard
Access the MetalLB metrics endpoint
MetalLB metrics reference
Troubleshooting MetalLB
Trademark Notice

Prerequisites

Please ensure that you have read the Installation documentation before proceeding.

Configure MetalLB Components

Configure the `MetalLB` custom resource's `spec.controllerNodeSelector`, `spec.controllerTolerations`, `spec.nodeSelector`, and `spec.speakerTolerations` fields after installing the Cluster Plugin. See Configure Component Placement for the configuration example and the relationship with `L2Advertisement.spec.nodeSelectors`.

To adjust the controller and speaker log level, set `spec.logLevel` on the same resource. See Adjust Log Level for supported values.

Configure an External IP Pool by using the web console

1. Go to **Administrator**.
2. In the left navigation bar, click **Networking** > **External IPs** > **External IP Pools**.
3. Click **Create External IP Pools**.
4. Refer to the following instructions to configure certain parameters.

Parameter	Description
Type	• 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. • 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.
IP Resources	Support input in CIDR and IP range formats. Click **Add** to support multiple entries, examples as follows: **CIDR**: `192.168.1.1/24`. **IP Range**: `192.168.2.1` ~ `192.168.2.255`.

Parameter	Description
Available Nodes	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. • **Node Name**: Select available nodes based on node names. • **Label Selector**: Select available nodes based on labels. • **Show Node Details**: View final available nodes in a list format. **Note**: • 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. • 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.
BGP Peers	Select BGP peers; please refer to BGP Peers 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 **Networking** > **External IPs** > **BGP Peer**.
3. Click **Create BGP Peer**.
4. Refer to the instructions below to configure the parameters.

Parameter	Description
Local AS Number	The AS number of the AS where the BGP-connected node resides.

Parameter	Description
	Note: 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.
Peer AS Number	The AS number of the AS where the BGP peer resides.
Peer IP	The IP address of the BGP peer, which must be a valid IP address capable of establishing a BGP connection.
Local IP	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.
Peer Port	The port number of the BGP peer.
BGP-Connected Node	The node that establishes the BGP connection. If this parameter is not configured, all nodes will establish BGP connections.
eBGP Multi-Hop	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.
RouterID	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**.

> Note: When you use BGP mode, allow TCP port 179 between each BGP peer and the MetalLB speaker nodes. If you configure a custom peer port, allow the configured port

> instead.

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

```yaml
# 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

```yaml
# 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-bgpadver
tisement.yaml
```

Monitor MetalLB

After you configure MetalLB, you can use the built-in dashboard and monitoring metrics to observe IP address allocation and BGP session status.

> Note: The built-in MetalLB dashboard is available in ACP 4.4.

View the built-in MetalLB dashboard

1. Go to **Operations Center**.
2. In the left navigation bar, click **Monitoring** > **Monitoring Dashboards**.
3. Search for **MetalLB**, and then open the dashboard.

The built-in MetalLB dashboard includes the following panels:

Panel	Description
Exposed Services	The number of LoadBalancer Services currently announced by MetalLB.
Available IPs	The number of IP addresses still available across all address pools.
Established BGP Sessions	The number of BGP sessions that are currently established.
Advertised Prefixes	The number of BGP prefixes currently advertised by MetalLB.
Pool Usage Trend	The IP address usage trend for selected address pools.
Selected Pool Capacity	The used and available IP address capacity for selected address pools.

Access the MetalLB metrics endpoint

MetalLB exposes controller and speaker metrics through monitor Services in the `metallb-system` namespace. You can access these metrics from inside the cluster to confirm whether the components expose metrics correctly.

Check the MetalLB monitor Services:

```
kubectl get svc -n metallb-system | grep monitor
```

Check the Service endpoints:

```
kubectl get endpointslice -n metallb-system -l kubernetes.io/service-name
=controller-monitor-service
kubectl get endpointslice -n metallb-system -l kubernetes.io/service-name
=speaker-monitor-service
```

The endpoint addresses use port `17472` by default.

Query the controller metrics endpoint from inside the cluster:

```
curl -s http://<controller-endpoint-address>:17472/metrics | grep '^metal
lb_'
```

Query a speaker metrics endpoint from inside the cluster:

```
curl -s http://<speaker-endpoint-address>:17472/metrics | grep '^metallb
_'
```

If the command returns MetalLB metrics, the component metrics endpoint is available.

MetalLB metrics reference

The built-in MetalLB dashboard uses the following metrics:

Metric	Source component	Description	Usage scenario
`metallb_speaker_announced`	Speaker	Indicates whether a LoadBalancer Service is announced from a speaker node.	Check w nodes announc LoadBala Services

Metric	Source component	Description	Usage scenario
`metallb_allocator_addresses_total`	Controller	Total number of IP addresses in address pools.	Calculate pool cap and avai IPs.
`metallb_allocator_addresses_in_use_total`	Controller	Number of IP addresses already allocated from address pools.	Calculate IPs and p usage tre
`metallb_bgp_session_up`	Speaker / BGP exporter	Indicates whether BGP sessions are established.	Check B session h
`metallb_bgp_announced_prefixes_total`	Speaker / BGP exporter	Number of BGP prefixes announced by MetalLB.	Check ro advertise status.

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
BGP not established	ASN mismatch or peer unreachable	Check `BGPPeer` spec and network routes
L2 not working	Wrong VLAN or ARP filtering	Use `arping` to verify broadcast reachability

To see more Troubleshooting MetalLB ↗

Trademark Notice

MetalLB™ is a trademark of The Linux Foundation.

Alauda is an independent vendor. This product is not affiliated with, endorsed by, or sponsored by The Linux Foundation. All trademarks are the property of their respective owners and are used here for identification purposes only.

Configure GatewayAPI Gateway

TOC

- Overview
- Prerequisites
- Gateway Basics
  - What is a Gateway
  - Gateway Exposure (Service Type)
    - LoadBalancer (**Recommended**)
    - NodePort
    - ClusterIP
  - Listener Configuration
    - Allow ListenerSets
    - Port and Protocol
    - AllowRouteNS
    - TLS Configuration
  - EnvoyProxy (Deployment Configuration)
    - Recommended Resource Sizing
    - Image Repository
- Create Gateway
  - Via Web Console
    - Listener Configuration
  - Via YAML

Complete Example with Multiple Listener Types
View Gateway Details
Listener and Route Reference
Hostname
Hostname Intersection Rule
Supported Route Kinds
Next Step
Trademark Notice

Overview

Configure a `Gateway` after the Envoy Gateway operator and `EnvoyGatewayCtl` are ready. A `Gateway` defines how traffic enters the gateway, while the companion `EnvoyProxy` controls how the underlying Envoy data plane is deployed.

In the recommended workflow, configure a `Gateway` after Envoy Gateway Operator and before Configure GatewayAPI Route.

To expose additional listeners owned outside the Gateway namespace, see Configure Gateway API ListenerSet.

Prerequisites

Please ensure that you have completed the following before proceeding:

1. Read Envoy Gateway Operator to understand the basic concepts and resource relationships
2. Install the Envoy Gateway operator and create an `EnvoyGatewayCtl`

> **NOTE**

Review the main Gateway concepts before creating a `Gateway`. If you are already familiar with these concepts, you can skip to the Create Gateway section.

Gateway Basics

What is a Gateway

A `Gateway` is the entry point for traffic entering your cluster. It defines how external requests are received and routed to your backend services. A `Gateway` mainly defines:

- **Listeners**: Define the ports, protocols, and hostnames the gateway listens on
- **GatewayClass**: Selects which gateway controller manages this `Gateway`
- **Infrastructure reference**: References an `EnvoyProxy` that controls how the underlying Envoy data plane is deployed

Gateway Exposure (Service Type)

Service Type configures how the gateway is exposed through the underlying Envoy Service. There are three modes: LoadBalancer, NodePort, and ClusterIP.

In YAML, this setting is configured in the companion `EnvoyProxy` resource at `.spec.provider.kubernetes.envoyService.type`.

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.

When using MetalLB, you can specify a static VIP through service annotations. In the Web Console, use the **Service Annotation** field:

```
metallb.universe.tf/address-pool: ADDRESS_POOL_NAME
# Or specify a specific IP directly
metallb.universe.tf/loadBalancerIPs: VIP_IP
```

For more details, see How To Specify a VIP When Using MetalLB.

NodePort

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

However, using NodePort has these disadvantages:

- 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

In the Gateway details page, when Service Type is NodePort, the listener list displays the NodePort column showing the assigned port numbers. You can also use the following command:

```
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 Configuration

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

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

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

Allow ListenerSets

A Gateway can accept additional listeners from ListenerSets in the same or another namespace. Configure `spec.allowedListeners.namespaces` to control which ListenerSet namespaces are accepted:

Value	Behavior
`None`	Reject all ListenerSets. This is the default.
`Same`	Accept ListenerSets in the Gateway namespace.
`Selector`	Accept ListenerSets in namespaces matching `selector`.
`All`	Accept ListenerSets in every namespace.

For `Selector`, configure a label selector under `.spec.allowedListeners.namespaces.selector`. For more information, see Configure GatewayAPI ListenerSet.

Port and Protocol

Each listener is configured with a port number and a protocol. Supported protocols are: HTTP, HTTPS, TCP, UDP, TLS.

AllowRouteNS

By default, Routes can only attach to a Gateway in the `Same` namespace. To allow cross-namespace routing, use the `Allowed Routes Namespace` field:

- `Same`: Allow Routes in the same namespace to attach to this listener.
- `All`: Allow Routes in any namespace to attach to this listener.

- `Selector`: Allow Routes in namespaces matched by the selector to attach to this listener.

In ACP, a project is identified by labels on namespaces, for example `cpaas.io/project: <project-name>`. If you want a listener to be used only by Routes from a specific project, use `Selector` and match the project label on the target namespaces.

The listener's `Allowed Routes Namespace` setting and its protocol together determine which listeners are available in the Route Web Console when you publish a route to a listener.

For more information, see attach to gateway created in other ns.

TLS Configuration

For HTTPS and TLS protocols, you need to configure TLS settings.

TLS Modes:

TLS Mode	Description	Certificate Required
Terminate	Envoy terminates TLS and decrypts traffic before forwarding to backend services	Yes, must select a TLS certificate
Passthrough	Envoy passes through encrypted TLS traffic to backend services without decryption	No

> **NOTE**
>
> - HTTPS protocol only supports **Terminate** mode
> - TLS protocol supports both **Terminate** and **Passthrough** modes
> - TLS listeners with **Passthrough** mode support TLSRoute
> - TLS listeners with **Terminate** mode support TCPRoute

Certificate Requirements:

- By default, you can only use secrets created in the same namespace

- The secret must be of type `kubernetes.io/tls` and contain `tls.crt` and `tls.key` keys
- For cross-namespace secrets, see use secret created in other ns

EnvoyProxy (Deployment Configuration)

Envoy Gateway uses the `EnvoyProxy` resource to control gateway deployment configurations. We recommend creating a dedicated `EnvoyProxy` resource for each Gateway and referencing it through the Gateway's `.spec.infrastructure.parametersRef` field.

When you create a `Gateway` from the Web Console by using a `GatewayClass` created by `EnvoyGatewayCtl`, the console automatically creates a companion `EnvoyProxy` resource with the same name and namespace.

When you create a `Gateway` by applying YAML, you are responsible for keeping the `Gateway` `.spec.infrastructure.parametersRef` and the referenced `EnvoyProxy` resource consistent.

This one-to-one mapping approach provides better isolation and more granular control over deployment configurations such as:

- Replicas
- Resource limits and requests
- Node selectors
- Service type and annotations
- Image repository

To limit the total Envoy data plane requests in a namespace, create an `EnvoyProxyResourceQuota` in that namespace. The quota evaluates this companion `EnvoyProxy` along with other `EnvoyProxy` resources in the namespace.

Recommended Resource Sizing

For typical HTTP and HTTPS north-south traffic, use the following values as starting points for a single `envoy-proxy` Pod. These recommendations apply to

`.spec.provider.kubernetes.envoyDeployment.container.resources`.

Peak QPS per `envoy-proxy` Pod	Recommended CPU	Recommended Memory
4,000	2 cores	1Gi
8,000	4 cores	2Gi
12,000	8 cores	4Gi

NOTE

- These values are starting points, not hard limits. Validate them against your actual traffic profile.
- TLS termination, enabled policies, route complexity, request and response sizes, and long-lived connections can all increase resource usage.
- When using multiple replicas, scale based on the observed QPS handled by each replica instead of only the total Gateway QPS.

Image Repository

The image repository is pre-configured with the default value for your cluster. Do not modify it unless necessary.

For other deployment configuration methods, see deployment-mode↗.

Create Gateway

Via Web Console

1. Navigate to `Alauda Container Platform -> Networking -> Gateway -> Gateways`
2. Click the `Create Gateway` button
3. On the `Create Gateway` page, select the `GatewayClass` created by your `EnvoyGatewayCtl`. In the recommended default example from Envoy Gateway Operator,

this is `envoy-gateway-operator-cpaas-default`.

The page displays the following configuration options:

Field	Description	YAML Path
Name	Gateway name	gateway: `.metadata.name` envoyproxy: `.metadata.name`
GatewayClass	Which GatewayClass to use	gateway: `.spec.gatewayClassName`
Service Type	Service Type	envoyproxy: `.spec.provider.kubernetes.envoyService.typ`
Service Annotation	Service annotations	envoyproxy: `.spec.provider.kubernetes.envoyService.ann`
Resource Limits	Deployment resource limits	envoyproxy: `.spec.provider.kubernetes.envoyDeployment.container.r`
Replicas	Deployment replicas	envoyproxy: `.spec.provider.kubernetes.envoyDeployment.`
Node Labels	Deployment node selectors	envoyproxy: `.spec.provider.kubernetes.envoyDeployment.nodeSelecto`
Listener	Listener	gateway: `.spec.listeners`

WARNING

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

NOTE

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

Listener Configuration

When creating or editing a listener, you can configure the following:

Field	Description	YAML Path
Name	The name of the listener	`.spec.listeners[].name`
Port	The port number the listener listens on	`.spec.listeners[].port`
Protocol	The protocol for the listener. Options: HTTP, HTTPS, TCP, UDP, TLS	`.spec.listeners[].protocol`
Hostname	Optional. The hostname for the listener	`.spec.listeners[].hostname`
Allowed Routes Namespace	Controls which namespaces can attach routes to this listener	`.spec.listeners[].allowedRoutes.namespaces.from`

HTTPS Protocol Configuration

When selecting **HTTPS** as the protocol:

Field	Description	YAML Path
Certificates	Required. Select the Kubernetes secret containing the TLS certificate.	`.spec.listeners[].tls.certificateRefs`

NOTE

- HTTPS protocol only supports **Terminate** mode
- Certificate selection is mandatory for HTTPS listeners
- You can only use secrets created in the same namespace by default

TLS Protocol Configuration

When selecting **TLS** as the protocol:

Configuration Fields

Field	Description	YAML Path
TLS Mode	Select the TLS mode. Options: Terminate, Passthrough. Default: Terminate	`.spec.listeners[].tls.mode`
TLS Certificate	Required only when TLS Mode is Terminate. Select the secret containing the TLS certificate.	`.spec.listeners[].tls.certificateRefs`

For TLS mode details, see TLS Configuration.

Via YAML

Replace `envoy-gateway-operator-cpaas-default` with the `GatewayClass` created by your own `EnvoyGatewayCtl` if you are not using the recommended default example.

The following minimal example creates an HTTP `Gateway` and a dedicated `EnvoyProxy`.

```yaml
apiVersion: gateway.networking.k8s.io/v1
kind: Gateway
metadata:
  name: demo
  namespace: 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
      allowedRoutes:
        namespaces:
          from: Same
---
apiVersion: gateway.envoyproxy.io/v1alpha1
kind: EnvoyProxy
metadata:
  name: demo
  namespace: demo
spec:
  provider:
    kubernetes:
      envoyService:
        type: ClusterIP
      envoyDeployment:
        replicas: 1
        container:
          imageRepository: registry.alauda.cn:60080/acp/envoyproxy/envoy
          resources:
            limits:
              cpu: '1'
              memory: 1Gi
            requests:
              cpu: '1'
              memory: 1Gi
    type: Kubernetes
```

Complete Example with Multiple Listener Types

If you need more listener types, use the following complete example:

```yaml
apiVersion: gateway.networking.k8s.io/v1
kind: Gateway
metadata:
  name: demo
  namespace: demo
spec:
  infrastructure: (1)
    parametersRef:
      group: gateway.envoyproxy.io
      kind: EnvoyProxy
      name: demo
  gatewayClassName: envoy-gateway-operator-cpaas-default (2)
  listeners: (3)
    - name: http
      port: 80
      hostname: a.com (4)
      protocol: HTTP (5)
      allowedRoutes:
        namespaces:
          from: Same (6)
    - name: https
      port: 443
      hostname: a.com
      protocol: HTTPS
      allowedRoutes:
        namespaces:
          from: Same
      tls: (7)
        mode: Terminate
        certificateRefs:
          - name: demo-tls
    - name: tls-passthrough
      port: 8443
      hostname: tls.example.com
      protocol: TLS
      allowedRoutes:
        namespaces:
          from: Same
      tls:
        mode: Passthrough
    - name: tls-terminate
      port: 9443
      hostname: secure.example.com
```

```
      protocol: TLS
      allowedRoutes:
        namespaces:
          from: Same
      tls:
        mode: Terminate
        certificateRefs:
          - name: demo-tls
    - name: tcp
      port: 8080
      protocol: TCP
      allowedRoutes:
        namespaces:
          from: Same
    - name: udp
      port: 8081
      protocol: UDP
      allowedRoutes:
        namespaces:
          from: Same
---
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. Reference to EnvoyProxy resource for deployment configuration
2. Replace `envoy-gateway-operator-cpaas-default` with your own `GatewayClass` if needed
3. `listeners` defines how traffic enters the gateway
4. `hostname` affects how host-based routing matches the listener
5. `protocol` determines which route kinds can attach to the listener
6. `allowedRoutes` controls which namespaces can attach routes
7. `tls` configures TLS termination or passthrough for HTTPS and TLS listeners
8. The `EnvoyProxy` name must match `.spec.infrastructure.parametersRef.name`
9. `envoyService.type` controls gateway exposure
10. Keep `imageRepository` unchanged unless necessary for your environment
11. `resources` configures the Envoy data plane resource limits and requests
12. Keep `provider.type` as `Kubernetes`

View Gateway Details

In the Gateway details page, the listener list shows the following information:

Column	Description
Name	The listener name
Protocol	The listener protocol (HTTP, HTTPS, TCP, UDP, TLS)
Port	The configured port number
NodePort	Displayed when Service Type is NodePort. Shows the assigned NodePort number for accessing the listener.

NOTE

When the Gateway Service Type is **NodePort**, the listener list displays an additional **NodePort** column. Use the NodePort value, not the service port, to access the gateway. For more details, see How to Get the Correct Port When Using NodePort.

Listener and Route Reference

Use the following rules when attaching `Route` resources to a `Gateway`.

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.

Hostname Intersection Rule

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., `api.example.com`)	All Route hostnames	Only requests with `api.example.com` are matched
Has hostname (e.g., `api.example.com`)	No hostnames	All Listener hostnames	Only requests with `api.example.com` are matched
Has hostname (e.g., `api.example.com`)	Has matching exact hostname	Exact match hostname	Only requests with `api.example.com` are matched

Listener Hostname	Route Hostnames	Intersection Result	Example
Has wildcard (e.g., `*.example.com`)	Has matching hostnames	Matching specific hostnames	Requests with `api.example.com` or `web.example.com` are matched
Has hostname (e.g., `api.example.com`)	Has non-matching hostnames	**No intersection** - 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 means the Route status becomes abnormal and traffic cannot be processed.

Supported Route Kinds

Each listener supports different route kinds based on its protocol:

Listener Protocol	Supported Route Kind
HTTP	HTTPRoute, GRPCRoute
HTTPS	HTTPRoute, GRPCRoute
TLS (`Passthrough` mode)	TLSRoute
TLS (`Terminate` mode)	TCPRoute
TCP	TCPRoute

Listener Protocol	Supported Route Kind
UDP	UDPRoute

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

Next Step

After the `Gateway` is ready, continue with Configure GatewayAPI Route. If you need advanced traffic control after routes are attached, continue with Configure GatewayAPI Policy. To limit Envoy data plane resources at the namespace level, see Configure EnvoyProxy Resource Quotas.

Trademark Notice

Envoy® is a registered trademark of The Linux Foundation.

MetalLB™ is a trademark of The Linux Foundation.

Alauda is an independent vendor. This product is not affiliated with, endorsed by, or sponsored by The Linux Foundation. All trademarks are the property of their respective owners and are used here for identification purposes only.

Configure GatewayAPI Policy

TOC

- Overview
- Prerequisites
- Policy Attachment Basics
- Policy Attachment Summary
- Create Policies in Web Console
- SecurityPolicy
  - Configuration Via Web Console
    - API Key Authentication
    - CORS Configuration
  - Configuration Via YAML
  - Reference
    - Features
    - How It Works
    - Notes
    - Official Documentation
- BackendTLSPolicy
  - Configuration Via Web Console
  - Configuration Via YAML
  - Reference
    - Features

Notes
Official Documentation
ClientTrafficPolicy
Configuration Via Web Console
Configuration Via YAML
Reference
Features
Notes
Official Documentation
BackendTrafficPolicy
Configuration Via Web Console
Configuration Via YAML
Reference
Features
Notes
Official Documentation
Related Tasks
Trademark Notice

Overview

This document explains how to configure policy resources after `Gateway` and `Route` resources are ready. Policies use the Policy Attachment pattern through `.spec.targetRefs` to attach additional traffic, security, and backend behavior to supported resources.

In the recommended workflow, this document comes after Configure GatewayAPI Route.

Envoy Gateway currently provides four policy types: `SecurityPolicy`, `BackendTLSPolicy`, `ClientTrafficPolicy`, and `BackendTrafficPolicy`.

Prerequisites

Please ensure that you have completed the following before proceeding:

1. Read Configure GatewayAPI Gateway and Configure GatewayAPI Route
2. Created the target resource that the policy will attach to, such as a `Gateway`, `Route`, or `Service`

Policy Attachment Basics

Policies attach to other resources through `.spec.targetRefs`.

By default, a policy can only attach to resources in the same namespace.

For `Gateway` targets, `sectionName` can be used to target a specific listener when the policy type supports it. For `Service` targets, `sectionName` refers to the Service port name.

Policy Attachment Summary

Policy Type	Purpose	Web Console Support	Gateway	HTTPRoute
SecurityPolicy	Authentication, authorization, CORS, and other security features	API Key Auth, CORS	✅ (listener name / `ALL`)	✅
BackendTLSPolicy	TLS configuration between Envoy and backend services	Supported	❌	❌
ClientTrafficPolicy	Client-facing timeout and	Timeout settings	✅ (listener name / `ALL`)	❌

Policy Type	Purpose	Web Console Support	Gateway	HTTPRoute
	connection behavior			
BackendTrafficPolicy	Backend timeout and connection behavior	Timeout settings	✅ (listener name / `ALL`)	✅

`sectionName` is used to target a specific listener on a `Gateway` or a specific port on a `Service`. When omitted or set to `ALL`, the policy applies to all listeners or ports.

For ListenerSet policy behavior and version limitations, see Configure Gateway API ListenerSet.

Create Policies in Web Console

All policy types are created from the same entry:

1. Navigate to `Alauda Container Platform -> Networking -> Gateway -> Policies`
2. Select the required value in the `Policy Type` dropdown
3. Click the `Create Policy` button

The following sections focus only on the fields that are specific to each policy type.

SecurityPolicy

Configuration Via Web Console

Common Fields (shared for all policies):

Field	Description	YAML Path
Policy Type	The type of policy to create	`.kind`
Attach To	The Gateway API resources this policy applies to. Supports Gateway, HTTPRoute, and GRPCRoute. When attaching to Gateway, you can optionally specify a listener name or select ALL listeners.	`.spec.targetRefs`

SecurityPolicy Specific Fields:

Field	Description	YAML Path
Authorization Type	The authentication/authorization method to use. Supports multiple selections: API Key Authentication, CORS Configuration	`.spec.apiKeyAuth`, `.spec.cors`

API Key Authentication

Field	Description	YAML Path
Secrets	Kubernetes secrets containing the API keys for authentication	`.spec.apiKeyAuth.credentialRefs`
Extract From	Specifies where to extract the API key from (HTTP headers or query parameters)	`.spec.apiKeyAuth.extractFrom`

CORS Configuration

Field	Description	YAML Path
Allow Origins	List of allowed origins for CORS requests	`.spec.cors.allowOrigins`
Allow Methods	List of allowed HTTP methods	`.spec.cors.allowMethods`
Allow Headers	List of allowed headers in CORS requests	`.spec.cors.allowHeaders`

Field	Description	YAML Path
Expose Headers	List of headers exposed to client in response	`.spec.cors.exposeHeaders`
Max Age	Cache duration for CORS preflight response	`.spec.cors.maxAge`
Allow Credentials	Whether credentials are allowed in CORS requests	`.spec.cors.allowCredentials`

Configuration Via YAML

```yaml
apiVersion: gateway.envoyproxy.io/v1alpha1
kind: SecurityPolicy
metadata:
  name: demo-security-policy
  namespace: demo
spec:
  targetRefs:
    - group: gateway.networking.k8s.io
      kind: HTTPRoute
      name: demo
  apiKeyAuth:
    credentialRefs:
      - group: ""
        kind: Secret
        name: demo
        namespace: demo
    extractFrom:
      - headers:
          - authorization
  cors:
    allowOrigins:
      - "https://example.com"
    allowMethods:
      - GET
      - POST
    allowHeaders:
      - "Content-Type"
    exposeHeaders:
      - "X-Custom-Header"
    maxAge: "1h"
    allowCredentials: true
```

Reference

SecurityPolicy is used to configure authentication, authorization, and other security-related features for your Gateway and Routes. It provides a declarative way to protect your services by validating incoming requests before they reach your backend applications.

```
            Gateway            Route
               |                 |
               +-- SecurityPolicy-+
Client ---> Envoy Listener --------------------> Backend Service
```

Features

- **Authentication**: Verify the identity of clients using various methods (API Key, JWT, OIDC, Basic Auth)
- **Authorization**: Control access to resources based on validated credentials
- **CORS Configuration**: Manage Cross-Origin Resource Sharing policies

How It Works

1. Create a SecurityPolicy with your desired authentication/authorization rules
2. Attach it to a specific Gateway, HTTPRoute, or GRPCRoute
3. Envoy Gateway validates incoming requests according to the policy
4. Valid requests are forwarded to backend services; invalid requests are rejected with appropriate HTTP status codes

Notes

1. The web console currently supports configuring **API Key Authentication** and **CORS**. For other authentication methods and advanced security features, you need to use YAML configuration.
2. Each Route can only be associated with one SecurityPolicy.
3. If a SecurityPolicy references a secret with no values, all requests to the attached route will be rejected with `401 Unauthorized`.
4. In the web console, by default, the `Extract From` field is set to `header` and the `Header Name` field is set to `authorization`.
5. You can view which policies are attached to a route by navigating to the Route's topology tab in the web console.

Official Documentation

- SecurityPolicy specification ↗
- API Key Authentication ↗

BackendTLSPolicy

Configuration Via Web Console

Common Fields:

Field	Description	YAML Path
Policy Type	The type of policy to create	`.kind`
Attach To	The Service this policy applies to. You must specify the Service name and port name (`sectionName`).	`.spec.targetRefs`

BackendTLSPolicy Specific Fields:

Field	Description	YAML Path
Hostname	Required. The SNI (Server Name Indication) used when Envoy connects to the backend service	`.spec.validation.hostname`
Subject Alternative Names	Optional. Used for backend HTTPS response validation. If not specified, the hostname value is used as default.	`.spec.validation.subjectAltNames`
Validation Type	The method for validating backend TLS certificates. Options: `CACertificateRefs` (use custom CA certificates), `WellKnownCACertificates` (use system CA certificates)	`.spec.validation`

CACertificateRefs Configuration:

Field	Description	YAML Path
CA Certificate Secret	Kubernetes secret containing the CA certificate. The secret must have a `ca.crt` key containing PEM-encoded TLS certificates.	`.spec.validation.cACertificateRefs`

NOTE

When creating or selecting a CA certificate secret:

- The secret type must be suitable for CA certificates
- The key must be `ca.crt`
- You can import a certificate file, which must start with `-----BEGIN CERTIFICATE-----` and end with `-----END CERTIFICATE-----`
- When importing an invalid certificate format, an error message "must contain PEM-encoded TLS certificates" will be displayed
- When selecting an existing secret without `ca.crt` key, an error message "must have ca.crt key" will be displayed

Configuration Via YAML

```yaml
apiVersion: gateway.networking.k8s.io/v1alpha2
kind: BackendTLSPolicy
metadata:
  name: demo-backend-tls-policy
  namespace: demo
spec:
  targetRefs:
    - group: ""
      kind: Service
      name: demo-backend
      namespace: demo
      sectionName: https-port
  validation:
    hostname: backend.example.com
    subjectAltNames:
      - backend.example.com
    cACertificateRefs:
      - group: ""
        kind: Secret
        name: backend-ca
        namespace: demo
```

Reference

BackendTLSPolicy controls the TLS configuration between Envoy Gateway and backend services. It allows you to configure:



- **SNI (Server Name Indication)**: The hostname used when establishing TLS connections to backends
- **Certificate Validation**: How to validate backend server certificates
- **CA Certificates**: Custom CA certificates for validating backend certificates

Features

- Configure TLS settings for connections to backend services
- Support for custom CA certificates or system well-known CA certificates
- SNI configuration for proper TLS handshake

Notes

1. The `sectionName` in `targetRefs` corresponds to the port name of the Service.
2. When using `WellKnownCACertificates`, the system's default CA certificates are used for validation.
3. The hostname is required and is used as the SNI value when Envoy connects to the backend.

Official Documentation

- BackendTLSPolicy specification ↗

ClientTrafficPolicy

Configuration Via Web Console

Common Fields:

Field	Description	YAML Path
Policy Type	The type of policy to create	`.kind`
Attach To	The Gateway this policy applies to. You can optionally specify a listener name or select ALL listeners.	`.spec.targetRefs`

Timeout Configuration (Options):

Field	Description	YAML Path
TCP Idle Timeout	The idle timeout for a TCP connection. Idle time is defined as a period in which there are no bytes sent or received on either the upstream or downstream connection. Default: 1 hour.	`.spec.settings.timeout.tcp.idleTimeout`
HTTP Request Received Timeout	The duration Envoy waits for the complete request reception. This timer starts upon request initiation and stops when either the last byte of the request is sent upstream or when the response begins. Default: 1 hour.	`.spec.settings.timeout.http.requestReceivedTimeout`
HTTP Idle Timeout	The idle timeout for an HTTP connection. Idle time is defined as a period in which	`.spec.settings.timeout.http.idleTimeout`

Field	Description	YAML Path
	there are no active requests in the connection. Default: unlimited.	
HTTP Stream Idle Timeout	The stream idle timeout defines the amount of time a stream can exist without any upstream or downstream activity. Default: 5 minutes.	`.spec.settings.timeout.http.streamIdleTimeout`

Configuration Via YAML

```yaml
apiVersion: gateway.envoyproxy.io/v1alpha1
kind: ClientTrafficPolicy
metadata:
  name: demo-client-traffic-policy
  namespace: demo
spec:
  targetRefs:
    - group: gateway.networking.k8s.io
      kind: Gateway
      name: demo
      sectionName: https
  settings:
    timeout:
      tcp:
        idleTimeout: "30m"
      http:
        requestReceivedTimeout: "60s"
        idleTimeout: "5m"
        streamIdleTimeout: "30s"
```

Reference

ClientTrafficPolicy controls the behavior of connections from clients to Envoy Gateway. It provides fine-grained control over:

```
            Gateway
               |
               +-- ClientTrafficPolicy
Client ---> Envoy Listener --------------------> Backend Service
          applies on the client-facing side ^
```

- **TCP Settings**: Connection-level timeout and keepalive settings
- **HTTP Settings**: Request/response timeouts and HTTP protocol behavior

Features

- Configure TCP connection idle timeouts
- Control HTTP request reception timeouts

- Set HTTP connection idle timeouts
- Configure HTTP stream idle timeouts

Notes

1. Timeout values are specified as duration strings (e.g., "30s", "5m", "1h").

Official Documentation

- ClientTrafficPolicy specification ↗

BackendTrafficPolicy

Configuration Via Web Console

Common Fields:

Field	Description	YAML Path
Policy Type	The type of policy to create	`.kind`
Attach To	The Gateway API resources this policy applies to. Supports Gateway, HTTPRoute, GRPCRoute, TCPRoute, UDPRoute, and TLSRoute. When attaching to Gateway, you can optionally specify a listener name or select `ALL` listeners. In YAML, this is configured through `sectionName` in `.spec.targetRefs`.	`.spec.targetRefs`

Timeout Configuration (Options):

Field	Description	YAML Path
TCP Connection Timeout	The timeout for network connection establishment,	`.spec.settings.timeout.tcp.connectionTimeout`

Field	Description	YAML Path
	including TCP and TLS handshakes. Default: 10 seconds.	
HTTP Connection Idle Timeout	The idle timeout for an HTTP connection. Idle time is defined as a period in which there are no active requests in the connection. Default: 1 hour.	`.spec.settings.timeout.http.connectionIdleTimeout`
HTTP Max Connection Duration	The maximum duration of an HTTP connection. Default: unlimited.	`.spec.settings.timeout.http.maxConnectionDuration`
HTTP Request Timeout	The time until the entire response is received from the upstream. Default: 15 seconds. Supports setting to unlimited.	`.spec.settings.timeout.http.requestTimeout`

Configuration Via YAML

```yaml
apiVersion: gateway.envoyproxy.io/v1alpha1
kind: BackendTrafficPolicy
metadata:
  name: demo-backend-traffic-policy
  namespace: demo
spec:
  targetRefs:
    - group: gateway.networking.k8s.io
      kind: HTTPRoute
      name: demo
  settings:
    timeout:
      tcp:
        connectionTimeout: "5s"
      http:
        connectionIdleTimeout: "30m"
        maxConnectionDuration: "1h"
        requestTimeout: "30s"
```

Reference

BackendTrafficPolicy controls the behavior of connections from Envoy Gateway to backend services. It provides fine-grained control over:

```
            Gateway / Route
                  |
                  +-- BackendTrafficPolicy
Client ---> Envoy Listener --------------------> Backend Service
                                  applies here ^
```

- **TCP Settings**: Connection establishment timeouts
- **HTTP Settings**: Connection durations, idle timeouts, and request timeouts

Features

- Configure TCP connection establishment timeouts
- Control HTTP connection idle timeouts
- Set maximum HTTP connection durations

- Configure HTTP request timeouts

Notes

1. Timeout values are specified as duration strings (e.g., "30s", "5m", "1h").
2. The `requestTimeout` field supports setting to "unlimited" to disable the timeout.

Official Documentation

- BackendTrafficPolicy specification ↗

Related Tasks

After policies are attached, continue with Tasks for Envoy Gateway for more operational examples and advanced configuration tasks.

Trademark Notice

Envoy® is a registered trademark of The Linux Foundation.

Alauda is an independent vendor. This product is not affiliated with, endorsed by, or sponsored by The Linux Foundation. All trademarks are the property of their respective owners and are used here for identification purposes only.

Configure GatewayAPI Route

TOC

- Overview
- Prerequisites
- Configuration
  - Configuration Via Web Console
    - Create HTTPRoute
    - Create TCP/UDP Route
    - Create GRPCRoute
    - Create TLSRoute
  - Configuration Via YAML
  - Route Field Reference
    - Publish to Listener
    - Backend
    - Hostnames
    - Rules
    - HTTPRoute Reference
    - GRPCRoute Match and Filter Reference
    - TLSRoute Reference
- View
  - Topology
- Next Step

Related Tasks

Trademark Notice

Overview

This document explains how to configure `Route` resources after a `Gateway` is ready. A `Route` attaches to one or more gateway listeners and defines how matching traffic is forwarded to backend services.

In the recommended workflow, this document comes after Configure GatewayAPI Gateway and before Configure GatewayAPI Policy.

In addition to create and update operations, this document also introduces the extra route viewing capabilities provided by the ACP Web Console.

Prerequisites

Please ensure that you have completed the following before proceeding:

1. Read Configure GatewayAPI Gateway to understand listeners, attachment rules, and `EnvoyProxy`
2. Create a `Gateway` that your `Route` will attach to

> **NOTE**
>
> This document first introduces each route type separately, then provides YAML examples, and finally explains the common route concepts in a shared reference section.

Configuration

A `Route` attaches to one or more listeners on a `Gateway`. The listener you can select depends on the route type, the listener protocol, and the listener's allowed route namespace settings.

Routes can also attach to an external `ListenerSet` listener. See Configure Gateway API ListenerSet.

Configuration Via Web Console

1. Navigate to `Alauda Container Platform -> Networking -> Gateway -> Routes`
2. Click the `Create Route` button
3. Select the route type (HTTPRoute, TCPRoute, UDPRoute, GRPCRoute, or TLSRoute)

Create HTTPRoute

Field	Description	YAML Path
Publish to Listener	publish to listener	`.spec.parentRefs`
Hostnames	hostnames	`.spec.hostnames`
Matches	matches	`.spec.rules[].matches`
Filters	filters	`.spec.rules[].filters`
Backend Instance	backend	`.spec.rules[].backendRefs`
Options	options	`.spec.rules[].filters`, `.spec.rules[].timeouts`, `.spec.rules[].retry`, `.spec.rules[].sessionPersistence`

Options Configuration

The Options field allows you to configure advanced traffic management settings:

Option	Description	YAML Path
Session Affinity	session persistence	`.spec.rules[].sessionPersistence`
Timeout	timeout settings	`.spec.rules[].timeouts`
Retry	retry policy	`.spec.rules[].retry`

Create TCP/UDP Route

Field	Description	YAML Path
Publish to Listener	publish to listener	`.spec.parentRefs`
Backend Instance	backend	`.spec.rules[].backendRefs`

Create GRPCRoute

Field	Description	YAML Path
Publish to Listener	publish to listener	`.spec.parentRefs`
Hostnames	hostnames	`.spec.hostnames`
Matches	grpc matches	`.spec.rules[].matches`
Filters	grpc filters	`.spec.rules[].filters`
Backend Instance	backend	`.spec.rules[].backendRefs`

Create TLSRoute

Field	Description	YAML Path
Publish to Listener	publish to listener	`.spec.parentRefs`
Hostnames	hostnames (optional)	`.spec.hostnames`

Field	Description	YAML Path
Backend Instance	backend	`.spec.rules[].backendRefs`

Configuration Via YAML

The following minimal example creates an `HTTPRoute` that attaches to the `https` listener of the `demo` `Gateway` and forwards matching traffic to a backend `Service`.

```yaml
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: demo-443
  namespace: demo
spec:
  hostnames:
    - example.com
  parentRefs:
    - group: gateway.networking.k8s.io
      kind: Gateway
      name: demo
      sectionName: https
  rules:
    - matches:
        - path:
            type: Exact
            value: /a
      backendRefs:
        - group: ''
          kind: Service
          name: echo-resty
          namespace: demo-space
          port: 80
          weight: 100
```

If you need more route types and advanced HTTPRoute options, use the following complete example:

```yaml
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
      timeouts: 7
        request: '30s'
        backendRequest: '10s'
      retry: 8
        codes:
          - 503
        attempts: 3
        backoff: '100ms'
      sessionPersistence: 9
        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
---
apiVersion: gateway.networking.k8s.io/v1alpha2
kind: GRPCRoute
metadata:
  name: grpc
  namespace: demo
spec:
  hostnames:
    - grpc.example.com
  parentRefs:
```

```
    - group: gateway.networking.k8s.io
      kind: Gateway
      name: demo
      sectionName: https
  rules:
    - matches:
        - method:
            type: service
            value: myservice
      filters:
        - type: RequestHeaderModifier
          requestHeaderModifier:
            set:
              - name: x-custom-header
                value: custom-value
      backendRefs:
        - group: ''
          kind: Service
          name: grpc-service
          port: 50051
---
apiVersion: gateway.networking.k8s.io/v1alpha2
kind: TLSRoute
metadata:
  name: tls
  namespace: demo
spec:
  hostnames:
    - tls.example.com
  parentRefs:
    - group: gateway.networking.k8s.io
      kind: Gateway
      name: demo
      sectionName: tls
  rules:
    - backendRefs:
        - group: ''
          kind: Service
          name: tls-backend
          port: 443
```

1 Hostnames

2 Publish to listener

3. Rules
4. Backend
5. Filters
6. Matches
7. Options
8. Timeout
9. Retry
10. Session Persistence

Route Field Reference

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 ↗

Publish to Listener

In Web Console

In the web console, you can select multiple listeners to publish the route to. The available listener candidates are filtered based on the following criteria:

- **User permissions**: You must have access to the gateway's namespace (the project must include this namespace).
- **Route namespace allowlist**: The gateway listener's allowed route namespaces must include the route's namespace.
- **Route kind matching**: The route's kind (HTTPRoute, GRPCRoute, etc.) must match the listener's allowed route kinds.

For more complex cross-namespace scenarios, please refer to attaching to a gateway created in another namespace.

Attach to a ListenerSet

The Web Console also lists accepted ListenerSet listeners. Select the ListenerSet and listener name when publishing a Route. The ListenerSet namespace, parent Gateway allowance, listener protocol, and listener `allowedRoutes` must permit the attachment.

In YAML

- 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.

To attach to a ListenerSet in YAML, set `parentRefs[].kind` to `ListenerSet`, set `name` and `namespace` to the ListenerSet, and set `sectionName` to the listener name:

```yaml
parentRefs:
  - group: gateway.networking.k8s.io
    kind: ListenerSet
    namespace: app-demo
    name: app-listeners
    sectionName: app-http
```

For more information, see Configure GatewayAPI ListenerSet.

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.

Hostnames

The `hostnames` field is supported by `HTTPRoute`, `GRPCRoute`, and `TLSRoute`. `TCPRoute` and `UDPRoute` do not use this field.

`hostnames` is a string array. It follows the Hostname Intersection Rules.

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)`.

The match structure is common across route types, but the supported match conditions depend on the route type. For example, `HTTPRoute` and `GRPCRoute` support different match condition sets.

Filters

Specifies transformations or modifications to apply to requests or responses.

The filter concept is common across route types, but the supported filter types depend on the route type.

HTTPRoute Reference

The following match conditions, filter types, and advanced options are used by `HTTPRoute`.

Match Condition Types

Object	Method	Value Types	Description	Value Requirements
Path				
	`Exact`	path (string)	Matches the URL path exactly and with case sensitivity. This means that an exact path match on /abc will only match requests to /abc, **NOT** /abc/, /Abc, or /abcd.	Must start with `/`, no consecutive `//`.
	`PathPrefix`	path (string)	Matches based on a URL path prefix split by /. Matching is case-sensitive and done on **a path element by element basis**. For example, the paths /abc, /abc/, and /abc/def would all match the prefix /abc, but the path /abcd would not.	Must start with `/`, no consecutive `//`.
	`RegularExpression`	path (string)	Regex Engine: RE2.	for example, `/api/v1/.*`.

Object	Method	Value Types	Description	Value Requirements
Header				
	`Exact`	name (header key) + value	Exact header value match.	
	`RegularExpression`	name (header key) + value	Regex Engine: RE2.	
QueryParam				
	`Exact`	name (param key) + value	Exact query parameter value match.	Parameter value: 1-1024 characters
	`RegularExpression`	name (param key) + value	Regex Engine: RE2.	
Method	-	method name	HTTP method match.	`GET`, `HEAD`, `POST`, `PUT`, `DELETE`, `CONNECT`, `OPTIONS`, `TRACE`, `PATCH`

Match Condition References

Condition Type	Official Documentation
Path	HTTPPathMatch ↗
Headers	HTTPHeaderMatch ↗
QueryParams	HTTPQueryParamMatch ↗
Method	HTTPMethod ↗

Filter Types

Type	Method	Value Types	Description
RequestHeaderModifier	`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)
ResponseHeaderModifier	`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)
RequestRedirect	`Scheme`	string	Scheme for Location header (http/https)

Type	Method	Value Types	Description
	`Hostname`	PreciseHostname	Hostname for Location header
	`ReplaceFullPath`	string	Replace entire request path
	`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

Type	Method	Value Types	Description
	`AllowMethods`	[]HTTPMethod	List of allowed HTTP methods
	`AllowHeaders`	[]string	List of allowed headers in CORS requests
	`ExposeHeaders`	[]string	List of headers exposed to client in response
	`MaxAge`	Duration	Cache duration for CORS preflight response
	`AllowCredentials`	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	HTTPHeaderFilter ↗
ResponseHeaderModifier	HTTPHeaderFilter ↗
RequestRedirect	HTTPRequestRedirectFilter ↗
URLRewrite	HTTPURLRewriteFilter ↗
CORS	HTTPCORSFilter ↗
RequestMirror	HTTPRequestMirrorFilter ↗
HTTPExternalAuthFilter	HTTPExternalAuthFilter ↗

Options

The Options section provides advanced traffic management capabilities for `HTTPRoute`, including timeout, retry, and session persistence settings.

Timeouts

Field	Description	YAML Path
Request Timeout	The maximum duration for the gateway to complete an HTTP response after receiving the full request from the client. Options: Default (uses default timeout, typically 15 seconds), Unlimited (sets to "0s" to remove timeout), Custom.	`.spec.rules[].timeouts.request`
Backend Request Timeout	The maximum duration for a single gateway-to-backend call, from when the gateway starts sending the request to when the full	`.spec.rules[].timeouts.backendRequest`

Field	Description	YAML Path
	backend response is received. Options: Default (uses implementation-specific default), Unlimited (sets to "0s"), Custom.	

NOTE

- The Request Timeout starts counting after the entire client request has been received and covers the complete transaction, which may include multiple backend calls if retries occur.
- Backend Request Timeout must be ≤ Request Timeout when specified.
- When selecting "Default", the field is set to nil (uses implementation default).
- When selecting "Unlimited", the field is set to "0s" (maximum possible value).

Field	Specification
`.spec.rules[].timeouts`	HTTPRouteTimeouts↗

Retry

Field	Description	YAML Path
Status Codes	HTTP status codes that trigger retry (e.g., 503, 502). Value range: 400-599.	`.spec.rules[].retry.codes`
Attempts	Number of retry attempts.	`.spec.rules[].retry.attempts`
Backoff	Wait time before retry (e.g., "100ms", "1s").	`.spec.rules[].retry.backoff`

NOTE

- By default, retries are **disabled**. If the retry field is not configured or is left empty, the gateway will NOT retry any failed requests.

- You must explicitly configure both retry attempts and retry conditions to enable retry functionality.
- When configuring retry in the web console, if you remove all retry configuration items, the field is set to nil.

Field	Specification
`.spec.rules[].retry`	HTTPRouteRetry ↗

Session Persistence

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

Field	Description	YAML Path
Type	Session persistence type. Options: Cookie, Header.	`.spec.rules[].sessionPersistence.type`
Session Name	The name of the cookie or header used for session tracking.	`.spec.rules[].sessionPersistence.sessionName`

Field	Specification
`.spec.rules[].sessionPersistence`	SessionPersistence ↗

GRPCRoute Match and Filter Reference

The following match conditions and filter types are used by `GRPCRoute`.

GRPCRoute Matches

`GRPCRoute` supports the following match types:

Object	Method	Value Types	Description
Method	-	type (service/method) + value	Matches gRPC method. Type can be `service` (matches service name) or `method` (matches method name).
Headers	`Exact`	name (header key) + value	Exact header value match.
	`RegularExpression`	name (header key) + value	Regex Engine: RE2.

GRPCRoute Filters

`GRPCRoute` only supports the `RequestHeaderModifier` filter:

Type	Method	Value Types	Description	Value Requiremer
RequestHeaderModifier	`Set`	name (string) + value (string)	Overwrites request header with given name and value	Max 16 item value: 1-409 chars
	`Add`	name (string) + value (string)	Adds header to request, appending to existing values	Max 16 item value: 1-409 chars
	`Remove`	[]string	Removes specified headers from request (case-insensitive)	Max 16 item

NOTE

`GRPCRoute` does not support Options such as timeout, retry, or session persistence.

TLSRoute Reference

The following behavior is specific to `TLSRoute`.

NOTE

- `TLSRoute` hostnames are optional. If the listener has a hostname but the `TLSRoute` does not, the `TLSRoute` automatically inherits the listener's hostname.
- `TLSRoute` can only attach to listeners with `TLS` protocol in `Passthrough` mode.

View

Topology

The following features are additional viewing capabilities provided by the ACP Web Console.

The topology tab provides a visual representation of the route and its associated resources. It displays all policies attached to the route, along with their dependent resources such as secrets referenced by `SecurityPolicy`.

This feature is currently only available for `HTTPRoute`.

Next Step

After routes are attached to listeners, continue with Configure GatewayAPI Policy if you need advanced traffic or security policies. For additional operational examples, see Tasks for Envoy Gateway.

Related Tasks

- How to attach to listener created in other namespace

Trademark Notice

Envoy® is a registered trademark of The Linux Foundation.

Alauda is an independent vendor. This product is not affiliated with, endorsed by, or sponsored by The Linux Foundation. All trademarks are the property of their respective owners and are used here for identification purposes only.

Configure GatewayAPI ListenerSet

TOC

- Overview
- Prerequisites
- Configuration
  - Configuration Via Web Console
  - Configuration Via YAML
- Attach a Route
- Policy behavior
- View ListenerSet Details
- Next Step
- Related Tasks
- Trademark Notice

Overview

Configure a `ListenerSet` to provide additional listeners for an existing `Gateway`. A ListenerSet is owned by its namespace and attaches to a Gateway through `spec.parentRef`.

ACP supports both listeners defined directly in `Gateway.spec.listeners` and listeners defined in a ListenerSet. A Gateway must explicitly allow ListenerSets; by default, it accepts none.

Prerequisites

Please ensure that you have completed the following before proceeding:

1. Read Configure GatewayAPI Gateway to understand Gateway listeners and listener attachment rules.
2. Create a Gateway and configure its ListenerSet namespace allowance.

Configuration

Configuration Via Web Console

1. Navigate to `Alauda Container Platform -> Networking -> Gateway -> ListenerSets`.
2. Click the `Create ListenerSet` button.
3. Select the ListenerSet namespace and its parent Gateway.
4. Add one or more listeners and configure their allowed Route namespaces.

The ListenerSet form contains the following fields:

Field	Description	YAML Path
Name	Name of the ListenerSet	`.metadata.name`
Namespace	Namespace that owns the ListenerSet	`.metadata.namespace`
Parent Gateway	Gateway that receives the listeners	`.spec.parentRef`

Field	Description	YAML Path
Listener	Port, protocol, hostname, TLS, and allowed Route namespaces	`.spec.listeners[]`

Configuration Via YAML

First configure the parent Gateway to allow ListenerSets from the required namespaces. See Allow ListenerSets on a Gateway.

Create a ListenerSet with `spec.parentRef` pointing to that Gateway:

```yaml
apiVersion: gateway.networking.k8s.io/v1
kind: ListenerSet
metadata:
  name: app-listeners
  namespace: app-demo
spec:
  parentRef:
    name: edge
    namespace: gateway-system
  listeners:
    - name: app-http
      protocol: HTTP
      port: 8080
      allowedRoutes:
        namespaces:
          from: Same
```

Listener names only need to be unique within the ListenerSet. A ListenerSet in another namespace is accepted only when the parent Gateway's `allowedListeners` rule includes that namespace.

Attach a Route

A Route can attach to a Gateway listener or a ListenerSet listener. Set the parent reference kind to `ListenerSet` and use `sectionName` for the selected listener:

```yaml
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: app-route
  namespace: app-demo
spec:
  parentRefs:
    - group: gateway.networking.k8s.io
      kind: ListenerSet
      namespace: app-demo
      name: app-listeners
      sectionName: app-http
  rules:
    - matches:
        - path:
            type: PathPrefix
            value: /app
      backendRefs:
        - name: app-service
          port: 80
```

The ListenerSet listener's `allowedRoutes` controls which Route namespaces may attach. The selected listener's protocol and hostname must also be compatible with the Route. For the complete Route workflow, see Attach a Route to a ListenerSet.

Policy behavior

Gateway-level policies without `sectionName` apply to the parent Gateway's merged listener configuration, including attached ListenerSet listeners.

On the ACP Envoy Gateway `v1.8.0` line, the following policies support a Gateway target with an optional `sectionName`, but do not support a direct `ListenerSet` target:

- `SecurityPolicy`
- `ClientTrafficPolicy`
- `BackendTrafficPolicy`
- `EnvoyExtensionPolicy`

`EnvoyPatchPolicy` targets a Gateway or GatewayClass and is not a ListenerSet-specific policy entry point. Use a Gateway-level policy when the behavior should cover the Gateway and its attached ListenerSet listeners. Use a Route-level policy when the behavior belongs to a particular Route.

View ListenerSet Details

In the ListenerSet details page, view the parent Gateway, listener configuration, and status conditions. When troubleshooting an attachment, check the Gateway's `status.attachedListenerSets`, the ListenerSet `Accepted` and `Programmed` conditions, and each listener's status.

Next Step

After the ListenerSet is ready, continue with Configure GatewayAPI Route to attach Routes. If you need traffic or security policies, continue with Configure GatewayAPI Policy.

Related Tasks

- Allow ListenerSets on a Gateway
- Attach a Route to a ListenerSet

Trademark Notice

Envoy® is a registered trademark of The Linux Foundation.

Alauda is an independent vendor. This product is not affiliated with, endorsed by, or sponsored by The Linux Foundation. All trademarks are the property of their respective owners and are used here for identification purposes only.

Configure EnvoyProxy Resource Quotas

TOC

Overview
Quota model
Create a quota
Configure EnvoyProxy requests
Admission behavior
Monitor namespace usage
Scope and limitations
Next step
Trademark Notice

Overview

Use `EnvoyProxyResourceQuota` to limit the total CPU and memory requests declared by `EnvoyProxy` resources in one namespace. The quota applies to the Envoy data plane configuration that an `EnvoyProxy` declares; it does not replace Kubernetes `ResourceQuota` and does not measure the generated Envoy Deployment directly.

This feature is available in ACP 4.5 and later. It is provided by the Envoy Gateway Operator and uses the `envoy-gateway.alauda.io/v1alpha1` API group.

NOTE

`Gateway` and `EnvoyProxy` are separate resources. We recommend one dedicated `EnvoyProxy` for each `Gateway`, with the `Gateway` referencing it through `.spec.infrastructure.parametersRef`. When using the ACP Web Console, the console can create the companion `EnvoyProxy` together with the `Gateway`; when applying YAML, create and reference both resources explicitly. The quota applies to the `EnvoyProxy` resources in the `Gateway` namespace. For more information, see Configure GatewayAPI Gateway.

Quota model

`EnvoyProxyResourceQuota` is a namespaced resource. A quota only applies to `EnvoyProxy` resources in the same namespace. The first version supports the following hard limits:

Resource	Meaning
`requests.cpu`	Total CPU requests for Envoy data plane replicas
`requests.memory`	Total memory requests for Envoy data plane replicas

The quota resource reports both the configured hard limit and the current namespace usage:

- `status.hard` mirrors `spec.hard`.
- `status.used` is the total usage counted for `EnvoyProxy` resources in the namespace.

If a namespace contains multiple quota objects, each object's `status.used` reports the same namespace total. Do not add the `status.used` values from multiple quota objects together. Each `EnvoyProxy` request must satisfy every quota object in its namespace.

Create a quota

Create a quota by applying a namespaced `EnvoyProxyResourceQuota` resource. The following example limits the namespace to 4 CPU cores and 8 GiB of memory requests for

Envoy data plane replicas:

```yaml
apiVersion: envoy-gateway.alauda.io/v1alpha1
kind: EnvoyProxyResourceQuota
metadata:
  name: gateway-quota
  namespace: tenant-a
spec:
  hard:
    requests.cpu: "4"
    requests.memory: 8Gi
```

```
kubectl apply -f gateway-quota.yaml
```

You can create more than one quota in a namespace when different policies must be checked at the same time. An `EnvoyProxy` is admitted only when its resulting usage is within every quota's hard limit.

Configure EnvoyProxy requests

The quota measures these fields in the `EnvoyProxy` resource:

EnvoyProxy field	Meani
`.spec.provider.kubernetes.envoyDeployment.replicas`	Numb Envoy plane
`.spec.provider.kubernetes.envoyDeployment.container.resources.requests.cpu`	CPU r for ea
`.spec.provider.kubernetes.envoyDeployment.container.resources.requests.memory`	Memo reque each r

For example, the following `EnvoyProxy` consumes `750m` CPU and `768Mi` memory from the quota:

```yaml
apiVersion: gateway.envoyproxy.io/v1alpha1
kind: EnvoyProxy
metadata:
  name: gateway-proxy
  namespace: tenant-a
spec:
  provider:
    type: Kubernetes
    kubernetes:
      envoyDeployment:
        replicas: 3
        container:
          resources:
            requests:
              cpu: 250m
              memory: 256Mi
```

The usage calculation is:

```
CPU usage = replicas × envoyDeployment.container.resources.requests.cpu
Memory usage = replicas × envoyDeployment.container.resources.requests.me
mory
```

When a field is omitted, the quota uses these defaults:

Field	Default
`replicas`	`1`
`requests.cpu`	`100m`
`requests.memory`	`512Mi`

The quota does not count resource limits. It also does not inspect the final Deployment generated by Envoy Gateway.

Admission behavior

When one or more quotas exist in a namespace, the Envoy Gateway Operator validates `EnvoyProxy` `CREATE` and `UPDATE` requests before they are stored:

1. The existing usage of the namespace is calculated from its `EnvoyProxy` resources.
2. For an update, the old object's usage is replaced by the new object's usage.
3. The resulting CPU and memory usage is checked against every quota in the namespace.
4. If any hard limit would be exceeded, the request is rejected. The error identifies the quota, resource, current usage, requested usage, and hard limit.

Deleting an `EnvoyProxy` reduces the reported `status.used` after the controller reconciles the quota. A namespace with no `EnvoyProxyResourceQuota` objects does not impose this EnvoyProxy-specific admission limit.

> **WARNING**
>
> Admission is a best-effort guardrail. Concurrent `CREATE` or `UPDATE` requests are evaluated against the usage observed at request time and are not atomically reserved. Plan headroom when strict isolation is required.

Monitor namespace usage

The quota status is the source of truth for the namespace-level total. List quotas to see the hard and used values:

```
kubectl get envoyproxyresourcequota -n tenant-a
```

The output includes these columns:

Column	Source	Meaning
`CPU Hard`	`.status.hard.requests.cpu`	CPU hard limit for this quota

Column	Source	Meaning
`CPU Used`	`.status.used.requests.cpu`	Total CPU request counted for EnvoyProxy resources in the namespace
`Memory Hard`	`.status.hard.requests.memory`	Memory hard limit for this quota
`Memory Used`	`.status.used.requests.memory`	Total memory request counted for EnvoyProxy resources in the namespace

For the complete values, inspect one quota resource as YAML:

```
kubectl get envoyproxyresourcequota gateway-quota -n tenant-a -o yaml
```

The controller updates `status.hard` and `status.used` asynchronously after a quota or `EnvoyProxy` changes. When a namespace has multiple quotas, display the namespace usage once and show each quota's hard limit separately; do not sum the repeated `CPU Used` or `Memory Used` values.

Scope and limitations

- This feature is available from ACP 4.5 onward.
- Only `requests.cpu` and `requests.memory` are supported.
- The quota is namespaced and only applies to `EnvoyProxy` resources in that namespace.
- The quota is independent of Kubernetes `ResourceQuota`; it does not read or update `ResourceQuota.status.used`.
- It does not limit resource `limits`.
- It does not track the final Envoy Deployment or enforce a Deployment admission policy.
- It does not support cross-namespace quotas, Gateway- or GatewayClass-specific groups, or label selectors.

Next step

After creating a quota, configure the corresponding Gateway and its dedicated `EnvoyProxy`. To understand how the Envoy Gateway Operator deploys these resources, see Envoy Gateway Operator.

Trademark Notice

Envoy® is a registered trademark of The Linux Foundation.

Alauda is an independent vendor. This product is not affiliated with, endorsed by, or sponsored by The Linux Foundation. All trademarks are the property of their respective owners and are used here for identification purposes only.

Configure ALB

WARNING

ALB is deprecated as of ACP 4.4. New ALB instances cannot be created, and ALB is no longer available in the web console. On a cluster upgraded from an earlier version, existing ALB instances keep running and can still be managed through the API, so that their traffic can be migrated to Envoy Gateway or Ingress NGINX. Remove the ALB instances once the migration is complete.

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

Trademark Notice

ALB

ALB is a custom resource that represents a load balancer. On a cluster upgraded from an earlier version, the alb-operator 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.

```yaml
# 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
`.spec.config.nodeSelector`	map[string]string	the node selector for the alb
`.spec.config.replicas`	int,optional default 3	the number of replicas for the alb
`.spec.config.resources.limits`	k8s container-resource,optional	limit of nginx container of alb
`.spec.config.resources.requests`	k8s container-resource,optional	request of nginx container of alb
`.spec.config.resources.alb.limits`	k8s container-resource,optional	limit of alb container of alb
`.spec.config.resources.alb.requests`	k8s container-resource,optional	request of alb container of alb
`.spec.config.antiAffinityKey`	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
`.spec.config.networkMode`	string: `host` or `container`, optional, default `host`	In `container` mode, the operator creates a LoadBalancer Service and uses its address as the ALB address.
`.spec.address`	string,required	you could manually specify the address of alb

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

project configuration

Field	Type
`.spec.config.projects`	[]string,required
`.spec.config.portProjects`	string,optional
`.spec.config.enablePortProject`	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

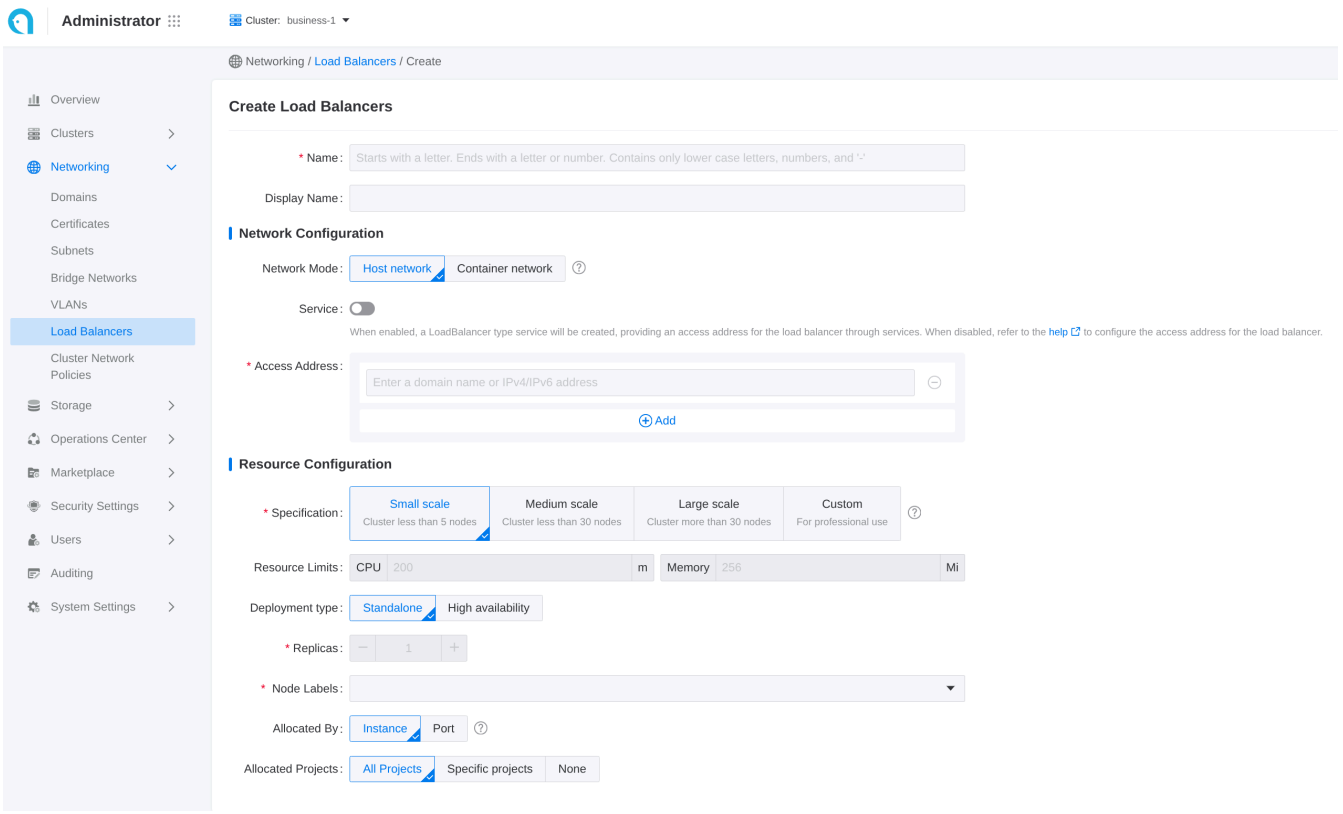
there are some global config which can be tweaked in alb cr.

- Ingress sync

Operation On ALB

Creating

Using the web console.



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	`.spec.address`
Allocated By	`Instance` 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

```yaml
# 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.

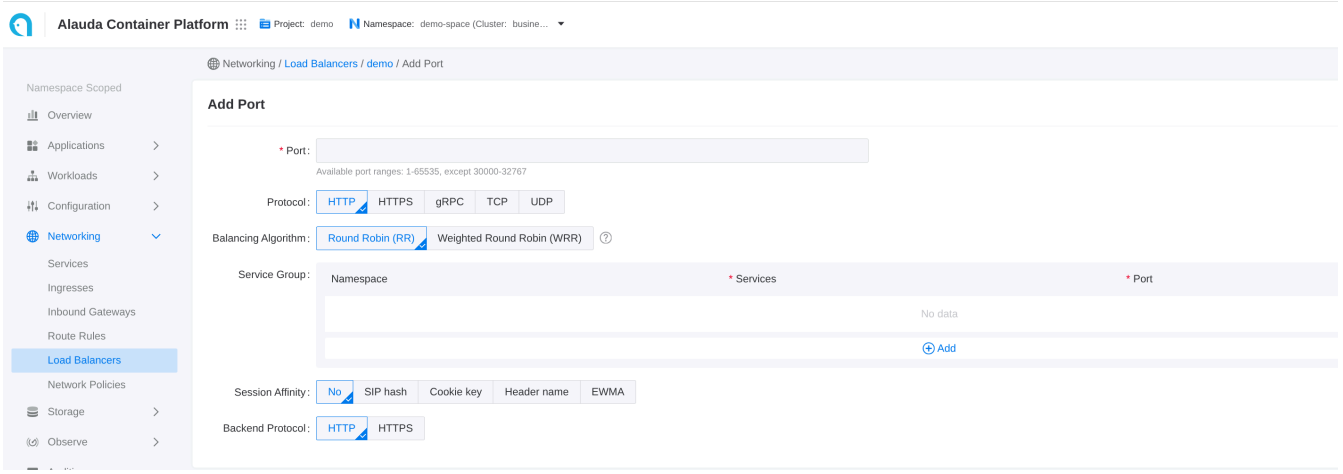
(5) certificate_name: for grpcs and https protocol which the default cert used, Format as `$secret_ns/$secret_name`.

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	`.spec.serviceGroup.session_affinity_policy`

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.

```yaml
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

```
dslx:
  - 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.	timeout
Redirect	Redirects incoming requests to a specified URL.	redirect
CORS	Enables Cross-Origin Resource Sharing (CORS) for the application.	cors
Header Modification	Allows modification of request or response headers.	header modification
URL Rewrite	Rewrites the URL of incoming requests before forwarding them.	url-rewrite
WAF	Integrates Web Application Firewall (WAF) for enhanced security.	waf
OTEL	Enables OpenTelemetry (OTEL) for distributed tracing and monitoring.	otel
Keepalive	Enables or disables the keepalive feature for the application.	keepalive

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

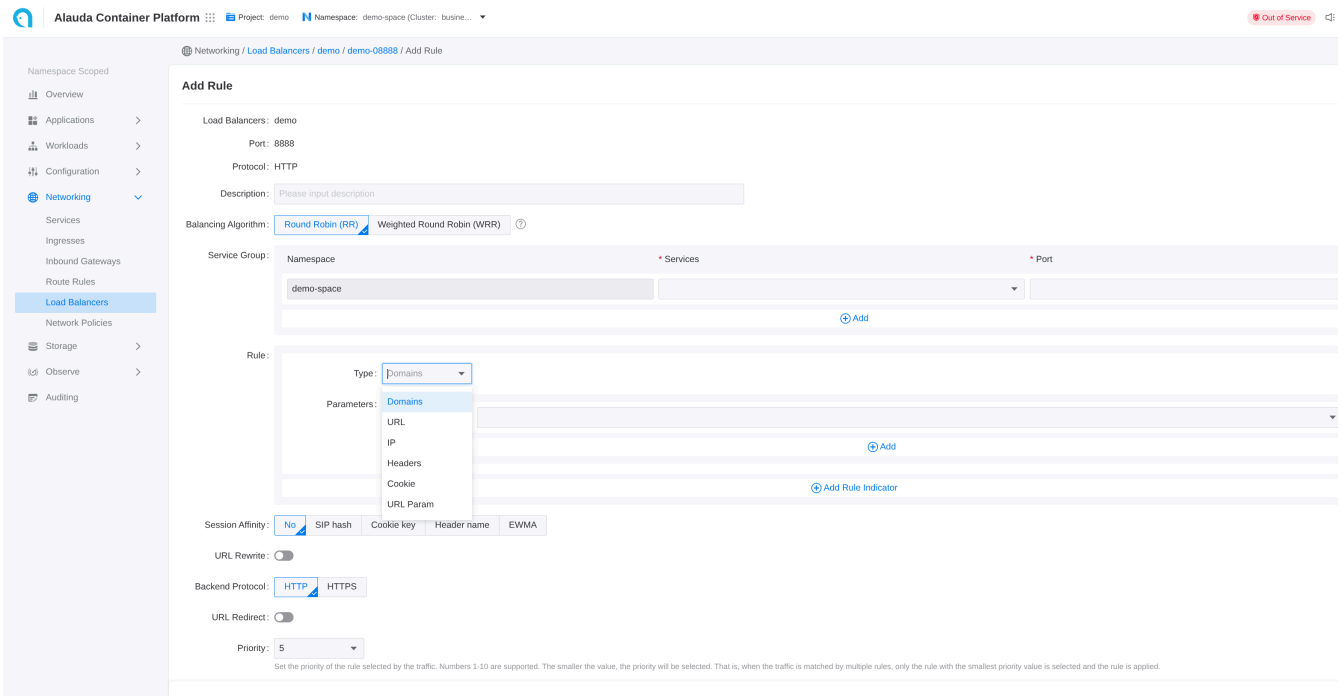
You can configure one or more services within a rule.

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



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-sy
stem/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.

Trademark Notice

Envoy® is a registered trademark of The Linux Foundation.

NGINX® is a registered trademark of F5, Inc. in the U.S. and in other countries.

Alauda is an independent vendor. This product is not affiliated with, endorsed by, or sponsored by The Linux Foundation or F5, Inc. All trademarks are the property of their respective owners and are used here for identification purposes only.

Configure Alauda Container Platform Node DNS Cache

TOC

Overview
Key Features
Important Notes
Installation
Install via Marketplace
How It Works
Architecture
Configuration
Network Policy Configuration
Known Risks and Temporary Workarounds
Risk 1: Host network port conflict
Risk 2: Node-level DNS dependency
Risk 3: Metrics collection through node IP
Trademark Notice

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.
5. **Cluster Upgrade via Rebuilding**: If the cluster is upgraded by rebuilding nodes (re-provisioning), kubelet configuration changes will be lost. To make the NodeLocal DNS configuration persistent across upgrades, you need to add the `--cluster-dns` parameter to `kubeletExtraArgs` in the following three places of the cluster template:
   - `KubeadmControlPlane` → `initConfiguration` → `nodeRegistration` → `kubeletExtraArgs`

- `KubeadmControlPlane` → `joinConfiguration` → `nodeRegistration` → `kubeletExtraArgs`
- `KubeadmConfigTemplate` → `template` → `spec` → `joinConfiguration` → `nodeRegistration` → `kubeletExtraArgs`

Add the following parameter to each of the above `kubeletExtraArgs` sections:

```
cluster-dns: "<NodeLocal_DNS_IP>"  # e.g., 169.254.20.10
```

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

1. Navigate to **Administrator** > **Marketplace** > **Cluster Plugins**.

2. Search for "**Alauda Container Platform Node DNS Cache**" in the plugin list.
3. Click **Install** to open the installation configuration page.
4. Configure the required parameters:

Parameter	Description	Example Value
IP	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

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

How It Works

Architecture

```
Pod → NodeLocal DNSCache → [Cache Hit] → Pod
            ↓
    [Cache Miss] → CoreDNS → Response → Cache & Pod
```

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:

```yaml
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
```

Known Risks and Temporary Workarounds

Treat this capability as Alpha for production readiness. It is recommended for test or pre-production environments. For production or production-like clusters, evaluate the following risks before enabling NodeLocal DNSCache. If these risks are not acceptable for your workloads, continue to use CoreDNS directly.

Before related parameters are exposed on the plugin installation page, use the following temporary workarounds only after validating them in the target environment.

Risk 1: Host network port conflict

The `node-cache` Pod runs in the host network namespace. Its health endpoint uses a host network port, which can conflict with workloads or node-level processes that use the same port. For example, the default health endpoint listens on port `8080`.

Before installation, confirm that workloads on the target nodes do not depend on the same host network port, for example port `8080`:

```
ss -lntp | grep ':8080'
```

Edit the NodeLocal DNSCache ConfigMap:

```
kubectl -n <namespace> edit cm <configmap-name>
```

Change the Corefile `health` port to the same unused port, for example:

```
health 127.0.0.1:18080
```

Edit the NodeLocal DNSCache DaemonSet:

```
kubectl -n <namespace> edit ds <daemonset-name>
```

Change the `node-cache` container probe port to the same port:

```
livenessProbe:
  httpGet:
    host: 127.0.0.1
    path: /health
    port: 18080
```

Wait for the DaemonSet rolling update to complete:

```
kubectl -n <namespace> rollout status ds/<daemonset-name>
```

If the current plugin version does not expose a health port parameter on the installation page, use a site-specific patch to keep the Corefile `health` port and the DaemonSet probe port consistent.

> **WARNING**
>
> **Important:** this is a temporary site patch, not persistent plugin configuration. Plugin upgrade, reinstall, or platform reconciliation can regenerate the ConfigMap and DaemonSet and overwrite your custom port setting. If you still need the non-default health port after those operations, reapply the patch before treating the plugin as ready.

Risk 2: Node-level DNS dependency

After NodeLocal DNSCache takes effect, Pods on a node use the local `node-cache` Pod for DNS resolution. If that Pod is unavailable during plugin upgrades, eviction, or crashes, DNS resolution for Pods on the same node may fail. Configure monitoring and alerting for `node-cache` readiness.

If DNS resolution has already failed and no fallback DNS server was configured in advance, temporarily point kubelet `cluster-dns` back to the CoreDNS ClusterIP and recreate the affected Pods.

You can configure kubelet `cluster-dns` with both the NodeLocal DNSCache IP and the CoreDNS ClusterIP. On each node that needs the change, edit the kubelet argument file:

```
vi /var/lib/kubelet/kubeadm-flags.env
```

Change `--cluster-dns` from the single NodeLocal DNSCache IP to both the NodeLocal DNSCache IP and the CoreDNS ClusterIP:

```
--cluster-dns=169.254.20.10,10.96.0.10
```

Restart kubelet after saving the file:

```
systemctl restart kubelet
```

This change only affects newly created Pods. Recreate the affected Pods so that their `/etc/resolv.conf` is regenerated, for example:

```
kubectl -n <namespace> rollout restart deployment/<deployment-name>
```

CoreDNS can be used as a fallback when NodeLocal DNSCache is unavailable, but this is not transparent failover. With the glibc resolver, DNS failover may wait for timeout across multiple `search` queries before using the next DNS server. musl-based images, such as Alpine Linux, usually do not have this delay because musl can query nameservers in parallel.

If the workload is sensitive to DNS resolution delay, tune resolver `timeout` and `attempts` for that workload. For example:

```
spec:
  dnsConfig:
    options:
      - name: timeout
        value: "1"
      - name: attempts
        value: "1"
```

WARNING

Important: editing kubelet files on nodes is also a temporary site change. Node rebuilds, re-provisioning, or any workflow that regenerates kubelet configuration can drop the multi-DNS setting completely. If you still need the CoreDNS fallback after those operations, reapply the kubelet change and then recreate the affected Pods again.

Risk 3: Metrics collection through node IP

If external monitoring cannot collect `node-cache` metrics through the node IP, edit the NodeLocal DNSCache ConfigMap:

```
kubectl -n <namespace> edit cm <configmap-name>
```

In the Corefile, remove the IP binding from the `prometheus` directive. Do not change the DNS service port.

```
prometheus 169.254.20.10:9253
```

```
prometheus :9253
```

Restart the `node-cache` Pods or roll the DaemonSet so that the updated Corefile takes effect.

> **WARNING**
>
> **Important:** this is also a temporary site patch, not persistent plugin configuration. If you patch the generated ConfigMap directly, plugin upgrade, reinstall, or platform reconciliation can overwrite the `prometheus` binding and bring the metric endpoint back to the previous default. Reapply the change after those operations if you still need external metrics collection through the node IP.

Trademark Notice

CoreDNS™ is a trademark of The Linux Foundation.

Kubernetes® is a registered trademark of The Linux Foundation in the United States and other countries.

Alauda is an independent vendor. This product is not affiliated with, endorsed by, or sponsored by The Linux Foundation. All trademarks are the property of their respective owners and are used here for identification purposes only.

Configure Alauda Container Platform Cluster DNS

TOC

Overview
Configuration
Host Alias
Node Selectors
Node Tolerations
Trademark Notice

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 Container Platform Cluster DNS**" 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) **Example**: `example.com test.example.com`

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	`NoSchedule`, `PreferNoSchedule`, or `NoExecute`

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

Trademark Notice

CoreDNS™ is a trademark of The Linux Foundation.

Alauda is an independent vendor. This product is not affiliated with, endorsed by, or sponsored by The Linux Foundation. All trademarks are the property of their respective owners and are used here for identification purposes only.

Network Observability

Network Observability provides continuous cluster-wide network flow collection and on-demand packet or flow capture for troubleshooting. It uses eBPF to inspect traffic behavior, identify performance issues, and investigate connectivity problems between workloads.

Note

Because Alauda Container Platform Network Observability Operator releases on a different cadence from Alauda Container Platform, the Alauda Container Platform Network Observability Operator documentation is now available as a separate documentation set at Alauda Container Platform Network Observability Operator ↗.

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

Trademark Notice

Tasks for Envoy Gateway

Tasks for Envoy Gateway

Overview

Prerequisites

Advanced Tasks

Related Documentation

More Configuration

Trademark Notice

Gateway API CRD versions

Gateway API CRD versions

Overview

ACP release mapping

Trademark Notice

Soft Data Center LB Solution (Alpha)

Soft Data Center LB Solution (Alpha)

Prerequisites

Procedure

Verification

Trademark Notice

Kube OVN

Understanding Kube-OVN CNI

Network Modes and Scope

Architecture and Components

Logical Network

How a Network Change Becomes a Flow

Traffic Paths

High Availability

Inspecting Components in Your Cluster

Next Steps

Preparing Kube-OVN Underlay

Usage Instructions

Terminology Explanation

Environment Requirements

Configuration Example

Automatic I

Procedure

Isolation Betwe

Kube-OVN Underlay + MetalLB

Overview

Prerequisites

Configuration Steps

Cluster Inte

Supports config
to enable inter-c

Prerequisites

Kub

clus

ster

pera

Configure Egress Gateway

Overview

Egress Gateway vs. Centralized Gateway

How Egress Gateway Works

Before You Begin

Configuration Workflow

Validated Capacity Guidance

Operations That May Interrupt Traffic

Additional Resources

Configuring Kube-OVN Network to Support P

Before You Begin

Installing Multus CNI

Create the Secondary Network

Create a Pod with Multiple Network Interfaces

Verify the Network Interfaces

Optional Configurations

Trademark Notice

Configure C

Using Label Se

Configure IPPool

Instructions

Precautions

Trademark Notice

Configure BGP

Installing `kube-ovn-speaker`

Publishing Pod and Subnet routes

Configure IPv4 Multicast for Kube-OVN Underlay Networks

Before You Begin

Enable Multicast Snooping

View the Configuration Result

Disable Multicast Snooping

Configure Endpoint Health Checker

Configure Endpoint Health Checker

Overview

Key Features

Installation

How It Works

How To Activate

Uninstallation

Trademark Notice

alb

Tasks for ALB

How To Set NodeSelector And Tolerations For alb-operator

How To Set NodeSelector And Tolerations For alb

Trademark Notice

Task: Migrate from OCP Route to GatewayAPI Route

Task: Migrate from OCP Route to GatewayAPI Route

Introduction

Prerequisites

Basic HTTP Route

Route Timeouts

HTTP Strict Transport Security (HSTS)

Cookie-Based Session Affinity

Path-Based Routing

Header Modification

Connection Limits

Rate Limiting

IP Allowlist/Blocklist

URL Rewrite

Cross-Namespace Route Admission

Default TLS Certificate for Ingress

TLS Re-encrypt with Custom CA

Edge Termination with Custom Certificate

TLS Passthrough

Feature Comparison Summary

Migration Strategy

Related Documentation

Trademark Notice

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
  - Limit a specific path
  - Configure the rejection status code
- 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
Trademark Notice

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	proxy-set-header ↗

action	link
remove header in request	set a empty header in request
set header in response	configuration-snippets ↗ with more-set-header ↗ directive
remove header in response	hide-headers ↗

URL Rewrite

rewrite ↗

HSTS (HTTP Strict Transport Security)

configure HSTS ↗

Rate Limiting

Ingress NGINX supports request-rate and concurrent-connection limits through Ingress annotations. For the complete annotation reference, see Rate Limiting ↗.

Limit a specific path

Annotations are applied to an Ingress object. If only one path, such as a login endpoint, must be limited, put that path in a separate Ingress object so that other paths are not limited by the same annotations.

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: login-rate-limit
  namespace: demo
  annotations:
    nginx.ingress.kubernetes.io/limit-rps: "10"
    nginx.ingress.kubernetes.io/limit-burst-multiplier: "2"
    nginx.ingress.kubernetes.io/limit-connections: "20"
spec:
  ingressClassName: demo-nginx
  rules:
    - host: app.example.com
      http:
        paths:
          - path: /auth/login
            pathType: Exact
            backend:
              service:
                name: app
                port:
                  number: 80
```

- `limit-rps` limits requests per second for each client IP handled by each Controller Pod.
- `limit-burst-multiplier` allows a burst in addition to the configured rate. The burst size is calculated from `limit-rps` and this multiplier.
- `limit-connections` limits concurrent connections for each client IP handled by each Controller Pod.

These limits are local to each Controller Pod. With multiple replicas, the effective cluster-wide threshold depends on traffic distribution and the replica count; it is not a single shared counter. Preserve the real client IP before relying on IP-based limits.

Configure the rejection status code

`limit-req-status-code` and `limit-conn-status-code` are Controller-wide settings. Configure them on the `IngressNginx` resource instead of editing the generated ConfigMap:

```yaml
apiVersion: ingress-nginx.alauda.io/v1
kind: IngressNginx
metadata:
  name: demo-nginx
  namespace: ingress-nginx-operator
spec:
  controller:
    config:
      limit-req-status-code: "429"
      limit-conn-status-code: "429"
```

Changing these values affects every Ingress handled by that Controller instance. Use a dedicated Controller when another application cannot accept the same rejection status code.

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

use the following yaml to deploy an ingress-nginx with default certificate

```yaml
apiVersion: ingress-nginx.alauda.io/v1
kind: IngressNginx
metadata:
  name: demo
spec:
  controller:
    extraArgs:
      default-ssl-certificate: $DEFAULT_CERT_NAMESPACE/$DEFAULT_CERT_NAME
```

please refer to 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:

```yaml
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
- **Tainted nodes**: If the selected nodes have a `NoSchedule` taint, configure a matching toleration for both Ingress-Nginx and MetalLB

How to configure

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

```yaml
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 routing traffic only to local pods
      annotations:
        metallb.universe.tf/address-pool: demo-pool  # Specify the MetalLB IP address pool to use
    nodeSelector:                           # Schedule pods only on nodes matching these labels. This selector must match the MetalLB L2Advertisement node selector
      dedicated: ingress
    tolerations:
      - key: dedicated
        operator: Equal
        value: ingress
        effect: NoSchedule
```

Important:

- The `nodeSelector` must match the nodes selected by the MetalLB `L2Advertisement` so that Ingress-Nginx pods are scheduled on nodes that can receive traffic from MetalLB.
- Configure the same node selection and toleration on the MetalLB custom resource: use `spec.nodeSelector` and `spec.speakerTolerations` for the speaker, and `spec.controllerNodeSelector` and `spec.controllerTolerations` for the controller. By default, MetalLB tolerates no additional taints.

For more details, see externalTrafficPolicy documentation ↗.

Trademark Notice

NGINX® is a registered trademark of F5, Inc. in the U.S. and in other countries.

MetalLB™ is a trademark of The Linux Foundation.

Alauda is an independent vendor. This product is not affiliated with, endorsed by, or sponsored by The Linux Foundation or F5, Inc. All trademarks are the property of their respective owners and are used here for identification purposes only.

Tasks for Envoy Gateway

TOC

- Overview
- Prerequisites
- Advanced Tasks
  - OpenTelemetry (OTel)
  - How To Attach to Listener Created In Other Namespace
  - How To Use a Certificate Created In Another Namespace
  - How To Use SSL Passthrough
  - How To Change the Minimum TLS Version
  - How To Specify NodePort When Using NodePort Service
  - How To Specify a VIP When Using MetalLB
  - How To Add Pod Annotations In Envoy Gateway
  - 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`
  - How To Use `hostNetwork` In `envoy-proxy`
    - Approach 1: Using Port Offset (Default, Recommended)
    - Approach 2: Using Privileged Ports Directly with useListenerPortAsContainerPort
  - How To Access a LoadBalancer VIP from Within the Cluster
  - How To Configure HSTS
  - How To Configure Custom Error Pages

Related Documentation

More Configuration

Trademark Notice

Overview

Use these advanced Envoy Gateway tasks after completing the main configuration path: operator, gateway, route, and policy.

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

1. **Modify standard Gateway API resources**: Directly edit `Gateway`, `HTTPRoute`, `TCPRoute`, `UDPRoute`, and other core resources.
2. **Attach policies via PolicyAttachment**: Use `SecurityPolicy`, `ClientTrafficPolicy`, `BackendTrafficPolicy`, and other policy resources to extend Gateway and Route behavior.
3. **Configure global settings**: Modify the `EnvoyGatewayCtl` to change `envoy-gateway` instance behavior or other global settings that affect all gateways.

Advanced tasks and special scenarios include cross-namespace routing, observability setup, deployment customization, and troubleshooting.

Prerequisites

1. Configure EnvoyGatewayCtl
2. Configure Gateway
3. Configure Route
4. Configure GatewayAPI Policy

Advanced Tasks

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 a Certificate Created In Another Namespace

To use a certificate created in another namespace, create a `ReferenceGrant` in the namespace where the certificate is stored. 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 the Minimum TLS Version

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` field in `ClientTrafficPolicy` is clienttlssettings↗. If you need to customize cipher suites in addition to the minimum TLS version, refer to the same upstream task and API reference.

How To Specify NodePort When Using NodePort Service

When using a NodePort service, Kubernetes assigns a NodePort value to each service port. When accessing the service through a node IP, use the assigned NodePort instead of the service port.

There are two approaches:

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.

```yaml
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 Specify a VIP When Using MetalLB

When using MetalLB as the LoadBalancer provider, you can specify a static VIP for the Gateway service through service annotations.

```yaml
apiVersion: gateway.envoyproxy.io/v1alpha1
kind: EnvoyProxy
metadata:
  name: demo
  namespace: demo
spec:
  provider:
    type: Kubernetes
    kubernetes:
      envoyService:
        type: LoadBalancer
        annotations: 1
          metallb.universe.tf/address-pool: production 2
          metallb.universe.tf/loadBalancerIPs: VIP_IP 3
```

1. Add MetalLB annotations in the `envoyService.annotations` field
2. Specify the address pool name to allocate IP from
3. Or specify a specific IP address (must be within the address pool range)

Available Annotations:

Annotation	Description
`metallb.universe.tf/address-pool`	Select the address pool to allocate IP from

Annotation	Description
`metallb.universe.tf/loadBalancerIPs`	Specify a specific IP address (supports multiple IPs, comma-separated)

NOTE

- The specified IP must be within a configured MetalLB address pool
- Make sure MetalLB is properly installed and configured before specifying VIPs
- For MetalLB configuration, see Configure MetalLB

How To Add Pod Annotations In Envoy Gateway

Set `annotations` in `.spec.provider.kubernetes.envoyDeployment.patch.value.spec.template.metadata.annotations` of `EnvoyProxy`.

```yaml
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:
                    your-annotation-key: 'your-annotation-value'
        container:
          imageRepository: registry.alauda.cn:60080/acp/envoyproxy/envoy
    type: Kubernetes
```

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

Update the `Subscription` resource.

```
# 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` resource.

```
# 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` resource.

```
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"
              }
            ]
          }
        }
      }
    }
  }
}'
```

How To Use `hostNetwork` In `envoy-proxy`

Using `hostNetwork: true` allows the Envoy proxy pods to use the host network namespace directly. This can be useful for:

- achieving better network performance
- accessing the gateway through the node IP directly

Considerations:

- Pods using hostNetwork will bind directly to the host's network interfaces
- Port conflicts may occur if multiple pods try to use the same port on the same node
- Security isolation is reduced as pods share the host's network namespace

- You should use `nodeSelector` or `affinity` rules to control pod placement and avoid port conflicts

There are two approaches to configure `hostNetwork`, depending on whether you want to use privileged ports (< 1024) directly:

Approach 1: Using Port Offset (Default, Recommended)

This is the default and recommended approach. Envoy Gateway automatically adds an offset of 10000 to privileged ports to avoid requiring special permissions.

Pros:

- No special permissions or capabilities required
- More secure because it runs as a non-root user without additional privileges
- Simpler configuration
- Works out of the box

Cons:

- Clients must use offset ports (10080, 10443)

Configuration:

```yaml
apiVersion: gateway.envoyproxy.io/v1alpha1
kind: EnvoyProxy
metadata:
  name: demo
  namespace: demo
spec:
  provider:
    type: Kubernetes
    kubernetes:
      envoyDeployment:
        patch:
          type: StrategicMerge
          value:
            spec:
              template:
                spec:
                  hostNetwork: true                    # Enable host netw
ork mode
                  dnsPolicy: ClusterFirstWithHostNet   # Required for pro
per DNS resolution
        pod:
          nodeSelector:                                # Recommended: con
trol pod placement to avoid conflict
            kubernetes.io/hostname: "demo"
```

Access:

- Port 80 → Access via `http://<node-ip>:10080`
- Port 443 → Access via `https://<node-ip>:10443`

Approach 2: Using Privileged Ports Directly with useListenerPortAsContainerPort

This approach allows Envoy to bind to privileged ports (< 1024) directly, such as port 80 and 443.

Pros:

- Can use standard ports (80 and 443) directly
- Better compatibility with clients expecting standard ports

Cons:

- Requires NET_BIND_SERVICE capability
- Slightly reduced security compared to port offset approach
- More complex configuration

Configuration:

```
apiVersion: gateway.envoyproxy.io/v1alpha1
kind: EnvoyProxy
metadata:
  name: demo
  namespace: demo
spec:
  provider:
    type: Kubernetes
    kubernetes:
      useListenerPortAsContainerPort: true  # Disable port offset
      envoyDeployment:
        patch:
          type: StrategicMerge
          value:
            spec:
              template:
                spec:
                  hostNetwork: true
                  dnsPolicy: ClusterFirstWithHostNet
                  containers:
                  - name: envoy
                    command:
                      - /usr/local/bin/envoy-with-cap  # use envoy with f
ilecap
                    securityContext:
                      capabilities:
                        add:
                        - NET_BIND_SERVICE  # Required for binding to pri
vileged ports
        pod:
          nodeSelector:                                # Recommended: con
trol pod placement to avoid conflict
            kubernetes.io/hostname: "demo"
```

Access:

- Port 80 → Access via `http://<node-ip>:80`
- Port 443 → Access via `https://<node-ip>:443`

How To Access a LoadBalancer VIP from Within the Cluster

By default, Envoy Gateway creates LoadBalancer services with `externalTrafficPolicy: Local`. This policy preserves client source IP addresses but has an important limitation: requests from cluster nodes without Envoy Gateway pods will fail because traffic is not forwarded to other nodes.

Solution 1: Use Service ClusterIP (Recommended for in-cluster access)

For applications running inside the cluster, use the service ClusterIP instead of the LoadBalancer VIP. This avoids the routing limitation entirely.

Solution 2: Change to Cluster Traffic Policy

If you need to access the LoadBalancer VIP from any cluster node, change `externalTrafficPolicy` to `Cluster`:

```
kubectl patch envoyproxy $GATEWAY_NAME -n $GATEWAY_NS --type='json' -p='[
  {"op": "replace", "path": "/spec/provider/kubernetes/envoyService/exter
nalTrafficPolicy", "value": "Cluster"}
]'
```

How To Configure HSTS

HTTP Strict Transport Security (HSTS) tells browsers to interact with a hostname only over HTTPS. ACP configures HSTS on a Gateway with `ClientTrafficPolicy` `.spec.headers.lateResponseHeaders`. Late response headers are applied by the Gateway listener after every other header modification, so the `Strict-Transport-Security` header is present on every response of the Gateway, including responses generated by Envoy itself

(for example the 404 returned when no route matches the request), which a route-level `ResponseHeaderModifier` filter cannot cover.

This feature is available in ACP 4.5 and later.

```
cat <<EOF | kubectl apply -f -
apiVersion: gateway.envoyproxy.io/v1alpha1
kind: ClientTrafficPolicy
metadata:
  name: hsts-policy
  namespace: default
spec:
  targetRef:
    group: gateway.networking.k8s.io
    kind: Gateway
    name: eg
  headers:
    lateResponseHeaders:
      set:
      - name: Strict-Transport-Security
        value: "max-age=31536000; includeSubDomains"
EOF
```

Verify the header on a normal response and on the Envoy-generated 404 of an unmatched host:

```
curl -sk -v --resolve www.example.com:443:<gateway-ip> https://www.exampl
e.com/ 2>&1 | grep -i strict-transport
curl -sk -v https://<gateway-ip>/ 2>&1 | grep -E "^< HTTP|strict-transpor
t"
```

Notes:

- Use `set` instead of `add` so that backends cannot weaken the header value. `addIfAbsent` and `remove` are also available in `lateResponseHeaders`.
- HSTS does not redirect HTTP traffic to HTTPS; it only tells browsers that already received the header over HTTPS to stay on HTTPS. Configure an HTTP-to-HTTPS redirect on the route as well if needed.

- Browsers cache the decision for `max-age` seconds and it cannot be undone from the server during that period. Validate with a small `max-age` (for example `300`) before rolling out one year.
- For the full API, see the official Envoy Gateway HTTP response headers documentation↗.

How To Configure Custom Error Pages

By default, the Gateway serves plain Envoy-generated error bodies (for example `no healthy upstream` on 503) and passes backend error responses through unchanged. Two mechanisms return custom error pages; both are available in ACP 4.5 and later:

- `BackendTrafficPolicy` `.spec.responseOverride[]` rewrites responses that match a status code.
- `HTTPRouteFilter` `.spec.directResponse` returns a fixed response for a route rule without contacting any backend.

`responseOverride` customizes error pages by status code. The body can be an inline string or the `response.body` key of a `ConfigMap`:

```
cat <<EOF | kubectl apply -f -
apiVersion: gateway.envoyproxy.io/v1alpha1
kind: BackendTrafficPolicy
metadata:
  name: error-page-policy
  namespace: default
spec:
  targetRef:
    group: gateway.networking.k8s.io
    kind: HTTPRoute
    name: backend
  responseOverride:
  - match:
      statusCodes:
      - type: Value
        value: 404
    response:
      contentType: text/html
      body:
        type: Inline
        inline: "<html><body><h1>Page not found</h1></body></html>"
  - match:
      statusCodes:
      - type: Value
        value: 500
    response:
      contentType: application/json
      body:
        type: ValueRef
        valueRef:
          group: ""
          kind: ConfigMap
          name: error-page-config
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: error-page-config
  namespace: default
data:
  response.body: '{"error": "Internal Server Error"}'
EOF
```

The optional `source` field of each rule controls which responses it rewrites: `All` (default) rewrites both backend and Envoy-generated responses, `Backend` only responses from backends, and `Local` only responses generated by Envoy itself, for example the 503 produced when a route matches but its backend has no ready endpoints. Use `Local` and `Backend` rules with the same status code to serve different pages for gateway errors and backend errors. A rule can also rewrite the status code (`response.statusCode`) or fetch the response from another URL (`redirect`).

`directResponse` answers a path without contacting any backend. Reference the filter from an HTTPRoute rule through an `ExtensionRef` ; the `HTTPRouteFilter` must live in the same namespace as the HTTPRoute, and the body size cannot exceed 4096 bytes:

```
cat <<EOF | kubectl apply -f -
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: backend
  namespace: default
spec:
  parentRefs:
  - name: eg
  hostnames:
  - www.example.com
  rules:
  - matches:
    - path:
        type: PathPrefix
        value: /maintenance
    filters:
    - type: ExtensionRef
      extensionRef:
        group: gateway.envoyproxy.io
        kind: HTTPRouteFilter
        name: maintenance-page
  - backendRefs:
    - name: backend
      port: 3000
---
apiVersion: gateway.envoyproxy.io/v1alpha1
kind: HTTPRouteFilter
metadata:
  name: maintenance-page
  namespace: default
spec:
  directResponse:
    contentType: text/html
    statusCode: 503
    body:
      type: Inline
      inline: "<html><body><h1>Under maintenance</h1></body></html>"
EOF
```

Verify the rewritten page:

```
curl -s -w '\n%{http_code}\n' --resolve www.example.com:80:<gateway-ip> h
ttp://www.example.com/missing-page
```

Notes:

- `responseOverride` only applies to responses that pass through the targeted HTTPRoute. A request that matches no route at all receives the default Envoy 404 and no rule applies to it.
- A direct response is generated by Envoy itself, so a `responseOverride` rule on the same HTTPRoute with a matching status code (for example 503 with `source: Local` or the default `source: All`) rewrites the direct response. Keep direct responses on a separate HTTPRoute when both mechanisms are used together.
- For the full API, see the official Envoy Gateway documentation for response override ↗ and direct response ↗.

Related Documentation

- Configure EnvoyGatewayCtl
- Configure Gateway
- Configure Route
- Configure GatewayAPI Policy

More Configuration

Please refer to EnvoyGateway Tasks ↗

Trademark Notice

Envoy® is a registered trademark of The Linux Foundation.

MetalLB™ is a trademark of The Linux Foundation.

Alauda is an independent vendor. This product is not affiliated with, endorsed by, or sponsored by The Linux Foundation. All trademarks are the property of their respective owners and are used here for identification purposes only.

Gateway API CRD versions

TOC

Overview

ACP release mapping

ACP 3.16.x - 4.1.x

ACP 4.2.1 - 4.2.x

ACP 4.3.x

ACP 4.4.x

Trademark Notice

Overview

ACP maintains and installs the standard Gateway API custom resource definitions (CRDs) as part of the platform lifecycle. The Envoy Gateway Operator manages Envoy Gateway extension CRDs in the `gateway.envoyproxy.io` API group; it does not own the standard Gateway API CRDs in the `gateway.networking.k8s.io` API group.

ACP uses the Gateway API CRDs from the Gateway API Experimental channel. This channel includes the standard Gateway API resources and experimental Gateway API resources such as `BackendTLSPolicy` and `ListenerSet`.

Do not install a second Gateway API CRD bundle in an ACP-managed cluster. Use the CRDs provided by the ACP release when creating or upgrading `Gateway`, Route, Policy, and ListenerSet resources.

ACP 3.16 was the first release with Gateway API CRDs. The version table below records the CRDs maintained by ACP for each release line.

ACP release mapping

Each release-line table lists one CRD per row. `Type` identifies whether the CRD comes from the Gateway API project or Envoy Gateway. `Version` contains the served versions, `Storage version` contains the version used to persist the resource, and `Bundle version` identifies the Gateway API CRD bundle. `N/A` means that the CRD is not part of the Gateway API bundle.

ACP 3.16.x - 4.1.x

Type	CRD	Version	Storage version	Bundle version
Gateway API	`GatewayClass`	v1alpha2, v1, v1beta1	v1beta1	v1.0.0
Gateway API	`Gateway`	v1alpha2, v1, v1beta1	v1beta1	v1.0.0
Gateway API	`GRPCRoute`	v1alpha2	v1alpha2	v1.0.0
Gateway API	`HTTPRoute`	v1alpha2, v1, v1beta1	v1beta1	v1.0.0
Gateway API	`ReferenceGrant`	v1alpha2, v1beta1	v1beta1	v1.0.0
Gateway API	`TCPRoute`	v1alpha2	v1alpha2	v1.0.0
Gateway API	`TLSRoute`	v1alpha2	v1alpha2	v1.0.0

Type	CRD	Version	Storage version	Bundle version
Gateway API	`UDPRoute`	v1alpha2	v1alpha2	v1.0.0
Gateway API	`BackendTLSPolicy`	v1alpha2	v1alpha2	v1.0.0

Changes:

- Gateway API CRDs are first introduced in ACP 3.16.
- This release line contains the initial standard Gateway API CRD set and no ACP Gateway API extension CRDs.

ACP 4.2.1 - 4.2.x

Type	CRD	Version	Storage version	Bundle version
Gateway API	`GatewayClass`	v1alpha2, v1, v1beta1	v1	v1.3.0
Gateway API	`Gateway`	v1alpha2, v1, v1beta1	v1	v1.3.0
Gateway API	`GRPCRoute`	v1alpha2, v1	v1	v1.3.0
Gateway API	`HTTPRoute`	v1alpha2, v1, v1beta1	v1	v1.3.0
Gateway API	`ReferenceGrant`	v1alpha2, v1beta1	v1beta1	v1.3.0
Gateway API	`TCPRoute`	v1alpha2	v1alpha2	v1.3.0
Gateway API	`TLSRoute`	v1alpha2	v1alpha2	v1.3.0
Gateway API	`UDPRoute`	v1alpha2	v1alpha2	v1.3.0

Type	CRD	Version	Storage version	Bundle version
Gateway API	`BackendTLSPolicy`	v1alpha2, v1alpha3	v1alpha3	v1.3.0
Gateway API	`XBackendTrafficPolicy`	v1alpha1	v1alpha1	v1.3.0
Gateway API	`XListenerSet`	v1alpha1	v1alpha1	v1.3.0

Changes:

- The Gateway API CRD bundle is updated to v1.3.0.
- Storage changes to `v1` for `GatewayClass`, `Gateway`, `GRPCRoute`, and `HTTPRoute`.
- `BackendTLSPolicy` adds `v1alpha3` and changes storage to `v1alpha3`.
- `XBackendTrafficPolicy` and `XListenerSet` are added.

ACP 4.3.x

Type	CRD	Version	Storage version	Bundle version
Gateway API	`GatewayClass`	v1alpha2, v1, v1beta1	v1	v1.4.1
Gateway API	`Gateway`	v1alpha2, v1, v1beta1	v1	v1.4.1
Gateway API	`GRPCRoute`	v1alpha2, v1	v1	v1.4.1
Gateway API	`HTTPRoute`	v1alpha2, v1, v1beta1	v1	v1.4.1
Gateway API	`ReferenceGrant`	v1alpha2, v1beta1	v1beta1	v1.4.1
Gateway API	`TCPRoute`	v1alpha2	v1alpha2	v1.4.1

Type	CRD	Version	Storage version	Bundle version
Gateway API	`TLSRoute`	v1alpha2, v1alpha3	v1alpha3	v1.4.1
Gateway API	`UDPRoute`	v1alpha2	v1alpha2	v1.4.1
Gateway API	`BackendTLSPolicy`	v1alpha2, v1alpha3, v1	v1	v1.4.1
Gateway API	`XBackendTrafficPolicy`	v1alpha1	v1alpha1	v1.4.1
Gateway API	`XListenerSet`	v1alpha1	v1alpha1	v1.4.1
Gateway API	`XMesh`	v1alpha1	v1alpha1	v1.4.1
Envoy Gateway	`Backend`	v1alpha1	v1alpha1	N/A
Envoy Gateway	`BackendTrafficPolicy`	v1alpha1	v1alpha1	N/A
Envoy Gateway	`ClientTrafficPolicy`	v1alpha1	v1alpha1	N/A
Envoy Gateway	`EnvoyExtensionPolicy`	v1alpha1	v1alpha1	N/A
Envoy Gateway	`EnvoyPatchPolicy`	v1alpha1	v1alpha1	N/A
Envoy Gateway	`EnvoyProxy`	v1alpha1	v1alpha1	N/A
Envoy Gateway	`HTTPRouteFilter`	v1alpha1	v1alpha1	N/A
Envoy Gateway	`SecurityPolicy`	v1alpha1	v1alpha1	N/A

Changes:

- The Gateway API CRD bundle is updated to v1.4.1.
- `BackendTLSPolicy` adds served `v1` and changes storage to `v1`.
- `TLSRoute` adds served `v1alpha3` and changes storage to `v1alpha3`.
- `XMesh` is added.
- Envoy Gateway CRDs are included as `v1alpha1` resources.

ACP 4.4.x

Type	CRD	Version	Storage version	Bundle version
Gateway API	`GatewayClass`	v1alpha2, v1, v1beta1	v1	v1.5.1
Gateway API	`Gateway`	v1alpha2, v1, v1beta1	v1	v1.5.1
Gateway API	`GRPCRoute`	v1alpha2, v1	v1	v1.5.1
Gateway API	`HTTPRoute`	v1alpha2, v1, v1beta1	v1	v1.5.1
Gateway API	`ReferenceGrant`	v1, v1alpha2, v1beta1	v1beta1	v1.5.1
Gateway API	`TCPRoute`	v1alpha2	v1alpha2	v1.5.1
Gateway API	`TLSRoute`	v1, v1alpha2, v1alpha3	v1	v1.5.1
Gateway API	`UDPRoute`	v1alpha2	v1alpha2	v1.5.1
Gateway API	`BackendTLSPolicy`	v1, v1alpha2, v1alpha3	v1	v1.5.1
Gateway API	`ListenerSet`	v1	v1	v1.5.1
Gateway API	`XBackendTrafficPolicy`	v1alpha1	v1alpha1	v1.5.1

Type	CRD	Version	Storage version	Bundle version
Gateway API	`XListenerSet`	v1alpha1	v1alpha1	v1.4.1
Gateway API	`XMesh`	v1alpha1	v1alpha1	v1.5.1
Envoy Gateway	`Backend`	v1alpha1	v1alpha1	N/A
Envoy Gateway	`BackendTrafficPolicy`	v1alpha1	v1alpha1	N/A
Envoy Gateway	`ClientTrafficPolicy`	v1alpha1	v1alpha1	N/A
Envoy Gateway	`EnvoyExtensionPolicy`	v1alpha1	v1alpha1	N/A
Envoy Gateway	`EnvoyPatchPolicy`	v1alpha1	v1alpha1	N/A
Envoy Gateway	`EnvoyProxy`	v1alpha1	v1alpha1	N/A
Envoy Gateway	`HTTPRouteFilter`	v1alpha1	v1alpha1	N/A
Envoy Gateway	`SecurityPolicy`	v1alpha1	v1alpha1	N/A

Changes:

- The standard Gateway API CRD bundle is updated to v1.5.1.
- `ListenerSet` is added.
- `ReferenceGrant` adds served `v1`.
- `TLSRoute` adds served `v1` and changes storage to `v1`.
- `BackendTLSPolicy` serves `v1` and uses it for storage.
- Envoy Gateway CRDs remain at `v1alpha1`.

For the configuration workflow, see Configure GatewayAPI Gateway, Configure GatewayAPI Route, and Configure GatewayAPI Policy.

Trademark Notice

Envoy® is a registered trademark of The Linux Foundation.

Alauda is an independent vendor. This product is not affiliated with, endorsed by, or sponsored by The Linux Foundation. All trademarks are the property of their respective owners and are used here for identification purposes only.

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

Trademark Notice

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.conntrack = 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
```

3. Restart using the `reboot` command.

4. Create a folder for the Keepalived configuration file.

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

5. 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 chec
k script
      check_timeout: 3  # optional, default 3: check script timeout per
iod
      name: "vip-1" # Identifier for this instance, can only contain al
phanumeric characters and hyphens, cannot start with a hyphen
      peer: [ "192.168.128.116", "192.168.128.75" ] # Keepalived node I
P, actual generated keepalived.conf will remove all IPs on the interfac
e.
      kube_lock:
        kubecfgs: # The kube-config list used by kube-lock will sequent
ially 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 control plane 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 t
he 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 sequent
ially 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::19
2:168:129:100" ]
      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 workload 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-certifi
cate-data | awk '{print $NF}' | base64 -d ) -noout -dates
```

7. Copy the `/etc/kubernetes/admin.conf` file from the three control plane 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/kubectl` from a control plane node of the workload cluster to the LB node.

    2. Execute `chmod +x /usr/bin/kubectl` 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/ke
epalived:/live/cfg build-harbor.alauda.cn/tkestack/keepalived:v3.16.0-b
eta.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 workload cluster ALBs.

```
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight        ActiveConn InAc
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>
```

Trademark Notice

NGINX® is a registered trademark of F5, Inc. in the U.S. and in other countries.

Alauda is an independent vendor. This product is not affiliated with, endorsed by, or sponsored by F5, Inc. All trademarks are the property of their respective owners and are used here for identification purposes only.

Kube OVN

Understanding Kube-OVN CNI

Network Modes and Scope
Architecture and Components
Logical Network
How a Network Change Becomes a Flow
Traffic Paths
High Availability
Inspecting Components in Your Cluster
Next Steps

Preparing Kube-OVN Underlay

Usage Instructions
Terminology Explanation
Environment Requirements
Configuration Example

Automatic I

Procedure
Isolation Betwe

Kube-OVN Underlay + MetalLB

Overview
Prerequisites
Configuration Steps

Cluster Inte

Supports config
to enable inter-c

Prerequisites

Kub
clus
ster
era

Configure Egress Gateway

Overview
Egress Gateway vs. Centralized Gateway
How Egress Gateway Works
Before You Begin
Configuration Workflow
Validated Capacity Guidance
Operations That May Interrupt Traffic
Additional Resources

Configuring Kube-OVN Network to Support P

Before You Begin
Installing Multus CNI
Create the Secondary Network
Create a Pod with Multiple Network Interfaces
Verify the Network Interfaces

Optional Configurations

Trademark Notice

Configure C

Using Label Se

Configure IPPool

Instructions

Precautions

Trademark Notice

Configure BGP

Installing `kube-ovn-speaker`

Publishing Pod and Subnet routes

Configure IPv4 Multicast for Kube-OVN Underlay Networks

Before You Begin

Enable Multicast Snooping

View the Configuration Result

Disable Multicast Snooping

Understanding Kube-OVN CNI

Kube-OVN connects Kubernetes Pods using Open Virtual Network (OVN) and Open vSwitch (OVS). It allocates Pod addresses, builds logical networks, and implements network policies and Service load balancing.

This page explains Kube-OVN architecture, logical networks, configuration updates, and common traffic paths. It also provides component inspection examples and links to configuration guides.

TOC

- Network Modes and Scope
- Architecture and Components
  - Control plane components
    - kube-ovn-controller
    - ovn-central
  - Node components
    - ovs-ovn
    - kube-ovn-cni
  - Monitoring and optional components
    - kube-ovn-pinger
    - kube-ovn-monitor
    - kube-ovn-speaker
  - Component deployment

Logical Network
Logical switches and subnets
IP address allocation and Subnet selection
The join switch and the node interface
Overlay encapsulation
Gateway modes
Network isolation and access control
How a Network Change Becomes a Flow
Traffic Paths
Pod to Pod on the same node
Pod to Pod across nodes
Pod to a Service
Pod to a node or an API server
Pod to an external network
High Availability
Inspecting Components in Your Cluster
List workloads
Locate Pods and containers
Inspect rollout state and events
Resource usage and limits
kubectl-ko
Next Steps

Network Modes and Scope

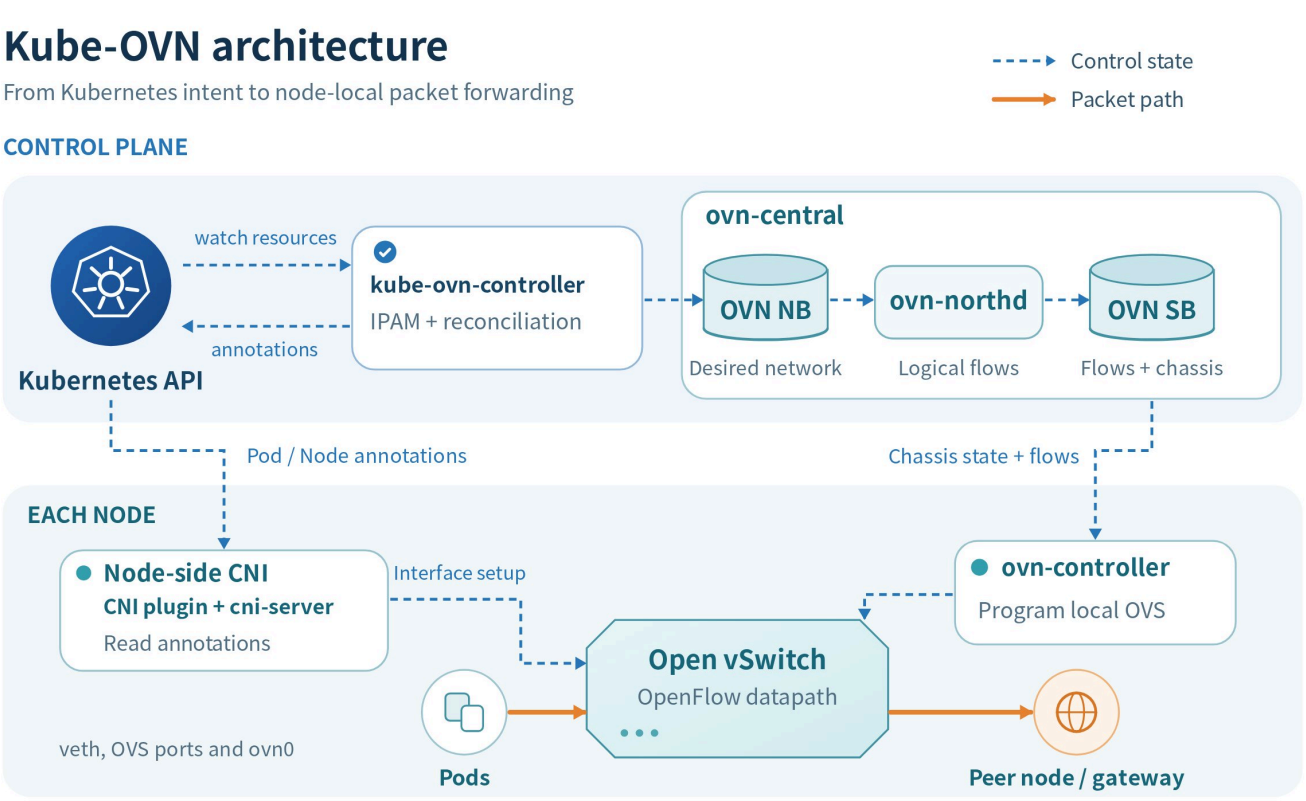
Kube-OVN supports overlay and underlay Subnets. The mode determines how Pod traffic reaches other nodes and the physical network.

Network mode	How it connects Pods	What to plan
Overlay	OVS transports cross-node Pod traffic over the node network using a tunnel. The default tunnel type is Geneve.	Pod CIDRs, node-to-node connectivity, tunnel traffic, and the MTU.
Underlay	A Subnet connects to the physical network through a VLAN and a ProviderNetwork (bridge network). Traffic on that physical network does not use the overlay tunnel.	Physical IP allocation, node interfaces, VLAN connectivity, and the physical gateway. See Prepare the underlay physical network.

The traffic-path diagrams in this page describe an overlay network in the default VPC with Geneve tunnels. The Service diagram also assumes that OVN Service load balancing is enabled for the controller and the source Subnet. Names and CIDRs shown as defaults can differ from your cluster configuration; check the Subnet settings and the installed Kube-OVN configuration with your administrator.

Architecture and Components

The control plane maintains the desired network and translates it into forwarding rules. Node components connect Pod interfaces and realize those rules in OVS. Monitoring and optional components provide health information and external integration.



OVN renders the logical network into local flows. OVS forwards packets on each node.

Control plane components

kube-ovn-controller

The `kube-ovn-controller` Deployment watches network-related Kubernetes resources, including Pods, Services, EndpointSlices, Nodes, NetworkPolicies, VPCs, Subnets, VLANs, and ProviderNetworks.

It allocates Pod addresses, writes logical ports and the required routes or policies to the OVN northbound database, and records assigned network information in Pod annotations for the node agent.

ovn-central

The `ovn-central` Deployment runs the OVN control plane services in its `ovn-central` container:

- `ovn-nb`: The northbound database stores the desired logical network, including switches, routers, ports, ACLs, and load balancers.
- `ovn-sb`: The southbound database stores logical flows, datapath and port bindings, and chassis state.

- `ovn-northd`: Translates northbound configuration into the logical flows and related records used in the southbound database.

For upstream implementation details, see the OVN architecture ↗. For the effects of component failures, see High availability.

Node components

ovs-ovn

The `ovs-ovn` DaemonSet runs an `openvswitch` container on each eligible Linux node. The container includes the local OVS database, `ovn-controller`, and `ovs-vswitchd`. The OVS database holds bridge and interface configuration, `ovn-controller` programs the forwarding rules, and `ovs-vswitchd` manages packet forwarding through the OVS datapath.

kube-ovn-cni

The `kube-ovn-cni` DaemonSet runs a `cni-server` container on each eligible Linux node. Its `install-cni` init container installs the `kube-ovn` CNI plugin and supporting binaries into the host's CNI binary directory (`/opt/cni/bin` by default), and writes the CNI configuration to `/etc/cni/net.d` by default. The container runtime invokes the `kube-ovn` plugin, which sends CNI requests to the local `cni-server`.

The node-side CNI plugin and `cni-server` work together to:

- Configure Pod interfaces, IP addresses, and routes during CNI ADD, and remove the corresponding attachments during CNI DEL.
- Attach Pod interfaces to OVS ports and maintain their local network configuration.

The `cni-server` daemon also maintains node networking by:

- Continuously reconciling host gateway filtering, NAT, and routing rules.
- Creating and configuring `ovn0` to connect the host network to the default VPC.

Additional node-network configuration has its own requirements. See Prepare the underlay physical network and Cluster Interconnection (Alpha).

Monitoring and optional components

kube-ovn-pinger

The `kube-ovn-pinger` DaemonSet runs a `pinger` container on each eligible Linux node to collect OVS status information and probe network connectivity and latency.

kube-ovn-monitor

The `kube-ovn-monitor` Deployment collects OVN status information and metrics. In the standard deployment, it runs on a node selected for OVN central services.

kube-ovn-speaker

This optional DaemonSet runs on nodes selected for BGP advertisement and publishes routes to the external network, allowing external access directly through Pod IPs. For installation and BGP options, see Configure BGP.

Component deployment

The following table describes a standard ACP installation with the OVN control plane in the same cluster. The Kubernetes workloads use the `kube-system` namespace by default. If your installation uses another namespace or a separately hosted OVN control plane, use its actual namespace and workload configuration.

Component	Kubernetes workload	Deployment location	Main container
`ovn-central`	Deployment	Nodes selected for OVN central services, with `kube-ovn/role=master` by default.	`ovn-central`
`kube-ovn-controller`	Deployment	Eligible Linux nodes selected by the scheduler; replicas are spread across nodes. It is not restricted to control-plane nodes by default.	`kube-ovn-controller`
`ovs-ovn`	DaemonSet	Each eligible Linux node.	`openvswitch`

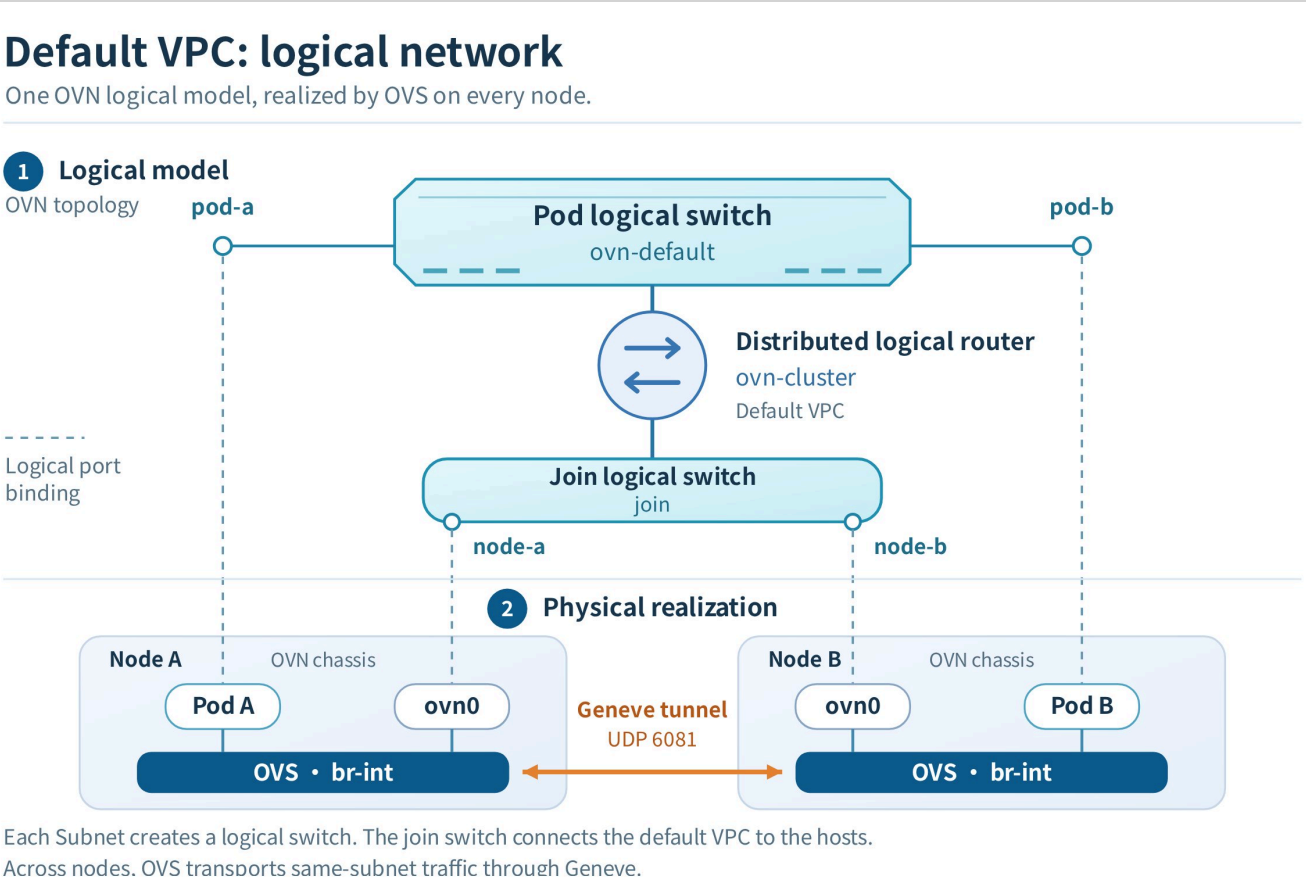
Component	Kubernetes workload	Deployment location	Main container
`kube-ovn-cni`	DaemonSet	Each eligible Linux node.	`cni-server`
`kube-ovn-pinger`	DaemonSet	Each eligible Linux node.	`pinger`
`kube-ovn-monitor`	Deployment	A node selected for OVN central services, with `kube-ovn/role=master` by default.	`kube-ovn-monitor`
`kube-ovn-speaker` (optional)	DaemonSet	Nodes selected for BGP advertisement, with `ovn.kubernetes.io/bgp=true` in the example manifest.	`kube-ovn-speaker`
`kubectl-ko`	Kubectl plugin	The administrator's workstation or another host with cluster access.	Not applicable

Node selectors, affinity, and tolerations determine which nodes are eligible. Deployment replica counts depend on the installation; a DaemonSet places a Pod on each eligible node. The optional speaker is installed separately as described in Configure BGP.

For commands and actual example outputs, see Inspecting components in your cluster.

Logical Network

OVN models the cluster network as logical switches and logical routers, and Kube-OVN maps its Kubernetes resources onto them. A VPC groups Subnets under its logical routing configuration. The following diagram shows the default VPC, including the `join` switch that connects it to the hosts. Custom VPCs have their own logical routers and require their own routing and external connectivity configuration. For resource fields, see the VPC API reference.



Each Subnet creates a logical switch. The join switch connects the default VPC to the hosts. Across nodes, OVS transports same-subnet traffic through Geneve.

In this default VPC overlay network, Pods attach to their Subnet's logical switch, the Subnet gateway connects that switch to the VPC logical router, and `join` connects the router to the nodes' `ovn0` ports. OVS implements the logical switching and routing on the nodes. Cross-node overlay traffic uses tunnels; traffic destined for a host or an external network leaves the overlay through the appropriate node interface. The sections below describe each part.

Logical switches and subnets

Each OVN-managed Subnet is backed by one logical switch. Pods attached to the Subnet become logical switch ports. A Namespace can have its own Subnet or share one with other Namespaces.

`kube-ovn-controller` allocates Pod IP addresses from the selected Subnet. The MAC address can be specified for the Pod or generated by IPAM. The controller records the assigned IP and MAC addresses, along with the Subnet CIDR, gateway, and logical switch, in Pod annotations. The node-side CNI components use this information, together with any additional route configuration, to configure the Pod interface and routes.

The default Subnet `ovn-default` carries the Pod network in a standard default VPC deployment. Custom VPCs have their own logical routers and Subnets. Attaching multiple

interfaces to one Pod is a separate configuration described in Pod multi-network interfaces.

IP address allocation and Subnet selection

Kube-OVN allocates an address from the Subnet selected for each Pod interface it manages. For the primary network, the platform's project or Namespace binding determines the available Subnets unless the Pod explicitly selects a Subnet. Without an explicit selection or binding, Pods use their VPC's default Subnet. For the platform allocation rules, see Configure Subnets.

When planning addresses, consider the following:

- **Available addresses:** The Subnet CIDR defines the address range. Reserved addresses in `excludeIps` are excluded from automatic allocation; reserving an address does not assign it to a Pod.
- **IP pools:** An IPPool restricts allocation to a set of addresses within a Subnet. Namespace or Pod settings select the pool. Running out of addresses in the eligible pools does not mean the Pod can use the remaining addresses elsewhere in the Subnet.
- **Address exhaustion:** If no eligible address is available, Pod network setup cannot complete. Check the Pod events and the available addresses in its selected Subnet or IPPool.
- **Pod replacement:** A replacement Pod is not guaranteed to receive the previous Pod's address through automatic allocation. Use a Service when applications need a stable access address.

Choose CIDRs and address families that match the cluster configuration. Check for conflicts with the node network, Service CIDRs, `join`, and external networks that the Pods must reach. For a secondary Pod interface, follow the separate Subnet and network attachment configuration in Pod multi-network interfaces.

The join switch and the node interface

Nodes are not part of a Pod Subnet. Kube-OVN connects each node to the overlay through a dedicated logical switch named `join` (CIDR `100.64.0.0/16` by default) and an OVS internal port named `ovn0` on the node.

`ovn0` connects the host network to the default VPC. `kube-ovn-controller` allocates the node an address from the `join` Subnet and records it in Node annotations; `kube-ovn-cni` reads those annotations to create and configure the interface. The Pod-to-node path below explains how traffic reaches a node or an API server endpoint on that node.

Overlay encapsulation

With the default Geneve network, OVS instances on different nodes are connected by tunnels. `ovn-controller` installs the forwarding rules, and OVS encapsulates cross-node overlay traffic in Geneve (UDP 6081) and sends it to the remote node's encapsulation address. The node network must allow this traffic between the participating nodes. See Configure the MTU for the encapsulation overhead this adds.

For underlay traffic, a VLAN Subnet connects to the physical network through a `localnet` port and the ProviderNetwork's node interfaces. Connectivity between underlay and overlay Subnets needs the appropriate routing configuration; see Automatic interconnection of underlay and overlay Subnets.

Gateway modes

For an overlay Subnet in the default VPC, `gatewayType` selects the nodes used for external egress. For Subnets in this VPC other than `join`, the controller sets an omitted `gatewayType` to `distributed`.

The logical router uses policy routes to direct that traffic to the selected node's address on the `join` switch:

- **Distributed**: each node acts as the gateway for the Pods running on it, so north-south traffic exits through the local node.
- **Centralized**: external egress is directed to selected gateway nodes. Multiple gateway nodes use active-backup mode by default; ECMP requires `spec.enableEcmp: true`.

These modes govern external egress; traffic between Pods or to a Service is not automatically sent through a centralized gateway. See the external access paths below and Configure a centralized gateway.

Network isolation and access control

A Namespace or Subnet assignment determines where workloads belong and how they receive addresses. It does not, by itself, deny traffic to workloads in other Namespaces or Subnets. Subnets in the same VPC can communicate through its logical router when routes and access-control rules allow it.

Use the following controls for their respective purposes:

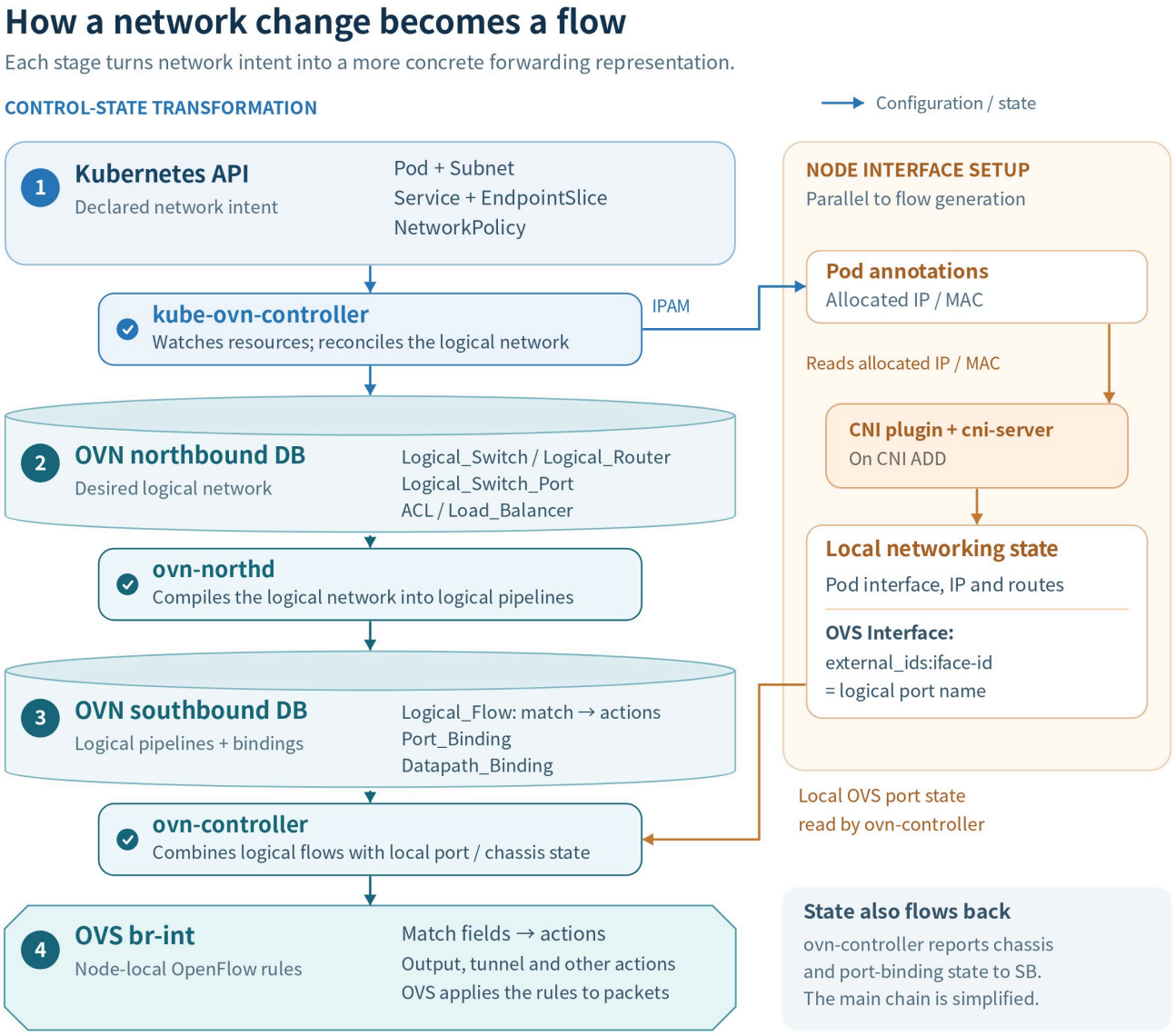
Control	Purpose
VPC	Groups Subnets under a separate logical routing configuration. Consider any configured routes, gateways, physical networks, and additional Pod interfaces when defining isolation between VPCs.
NetworkPolicy	Selects Pods and controls their ingress or egress traffic. Policies can use Pod and Namespace selectors or IP ranges.
Subnet isolation and ACLs	Apply access-control rules at the Subnet's logical network. Check these rules as well as NetworkPolicy when diagnosing connectivity.

For standard NetworkPolicy, ingress and egress isolation are evaluated separately. When a direction is isolated, the traffic must be allowed by a policy selecting the Pod in that direction; a connection must satisfy the applicable rules at both ends. If cluster-wide policies or Subnet ACLs are configured, their actions and evaluation order also affect the result. See Network Policy and Network Policy APIs.

Pods using `hostNetwork` share the node network and do not have an ordinary Pod network interface on the Subnet. Do not use ordinary Pod NetworkPolicy assumptions to describe their isolation; check the node's access controls and the documented policy limitations.

How a Network Change Becomes a Flow

Kubernetes resources describe the desired network. Kube-OVN and OVN translate that configuration into progressively more detailed representations until each node has the OpenFlow rules needed to forward packets.



The arrows show configuration and state updates. Packet forwarding takes place in OVS on the nodes.

1. **Kubernetes resources to logical network objects.** `kube-ovn-controller` watches resources such as Pods, Subnets, Services, and NetworkPolicies. It reconciles them into the OVN northbound database (`ovn-nb`): logical switches, routers, ports, ACLs, and load balancers describe the intended connectivity.
2. **Logical network objects to logical flows.** `ovn-northd` reads the northbound configuration and builds the logical processing pipeline in the OVN southbound database (`ovn-sb`). `Logical_Flow` records describe match conditions and actions; `Datapath_Binding` and `Port_Binding` records identify the logical datapaths and ports.
3. **Logical flows to node-local OpenFlow rules.** The `ovn-controller` on each node combines southbound information with its chassis identity and local OVS interface state. It

installs OpenFlow rules in the local OVS integration bridge, `br-int`, so OVS can implement the logical network on that node.

Pod interface setup supplies the local state needed by this process. During Pod creation, the node-side CNI plugin and `cni-server` work together to configure the Pod interface and OVS port using the assigned IP and MAC addresses. The OVS interface's `external_ids:iface-id` identifies the corresponding OVN logical port, allowing `ovn-controller` to associate that port with the node. `ovn-controller` also reports chassis and port-binding state back to the southbound database.

Administrators can inspect these layers with kubectl-ko.

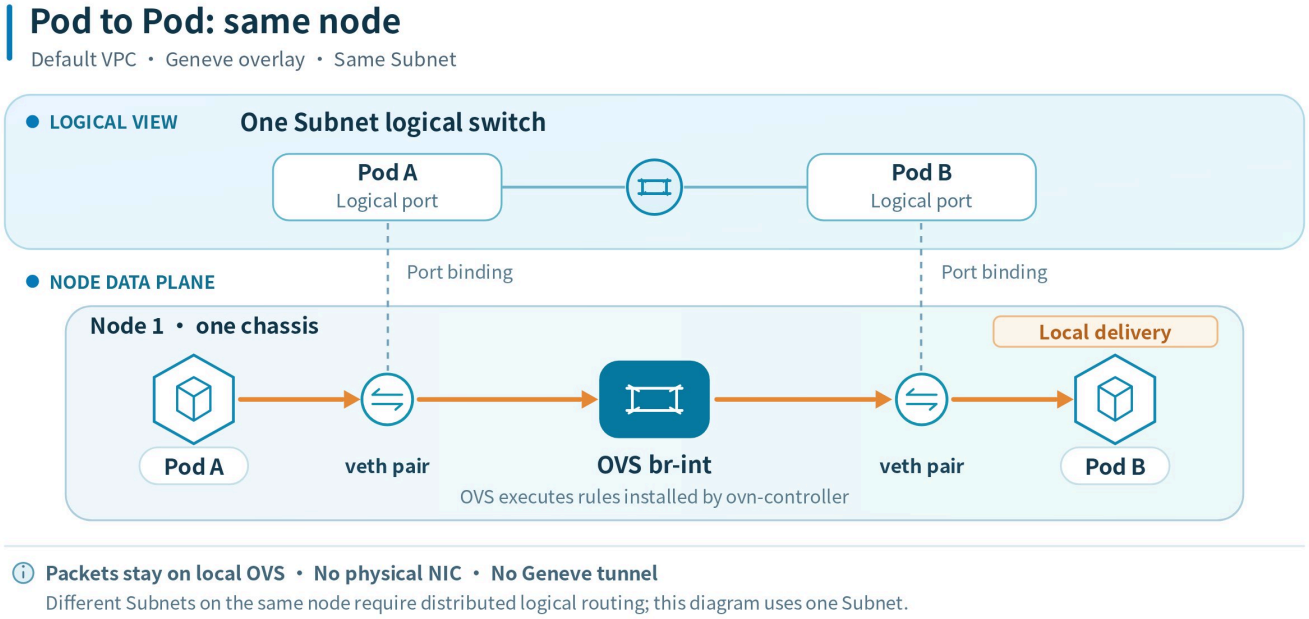
Traffic Paths

The diagrams below describe the default VPC overlay network with Geneve tunnels. OVS executes the logical switching, routing and load-balancing rules installed by `ovn-controller`. NetworkPolicy and Subnet ACLs can allow or block traffic on these paths.

Underlay networks and custom VPC egress have different topologies. Use the network mode and scope of your deployment when interpreting these diagrams.

Pod to Pod on the same node

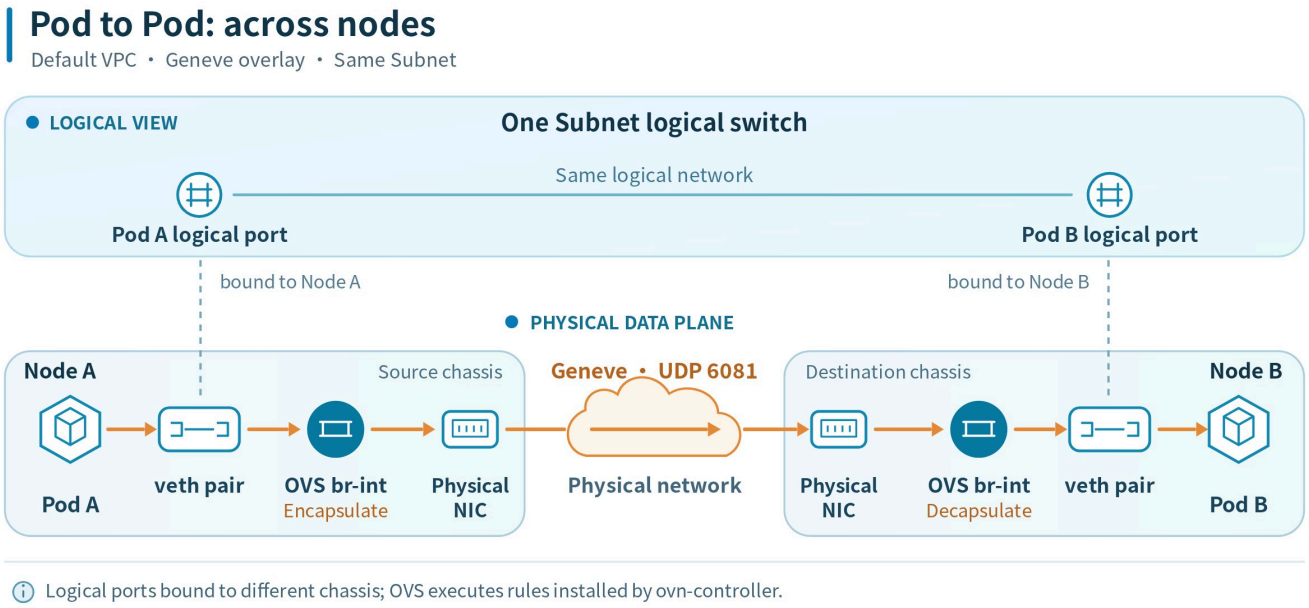
For two Pods in the same Subnet on one node, the packet travels from the source Pod interface through the local OVS bridge `br-int` to the destination Pod interface. The logical switch provides the forwarding rules; the packet does not use a physical NIC or a Geneve tunnel.



When the Pods belong to different Subnets in the same VPC, the VPC logical router provides the routing rules. OVS still processes the traffic locally when both Pods are on the same node.

Pod to Pod across nodes

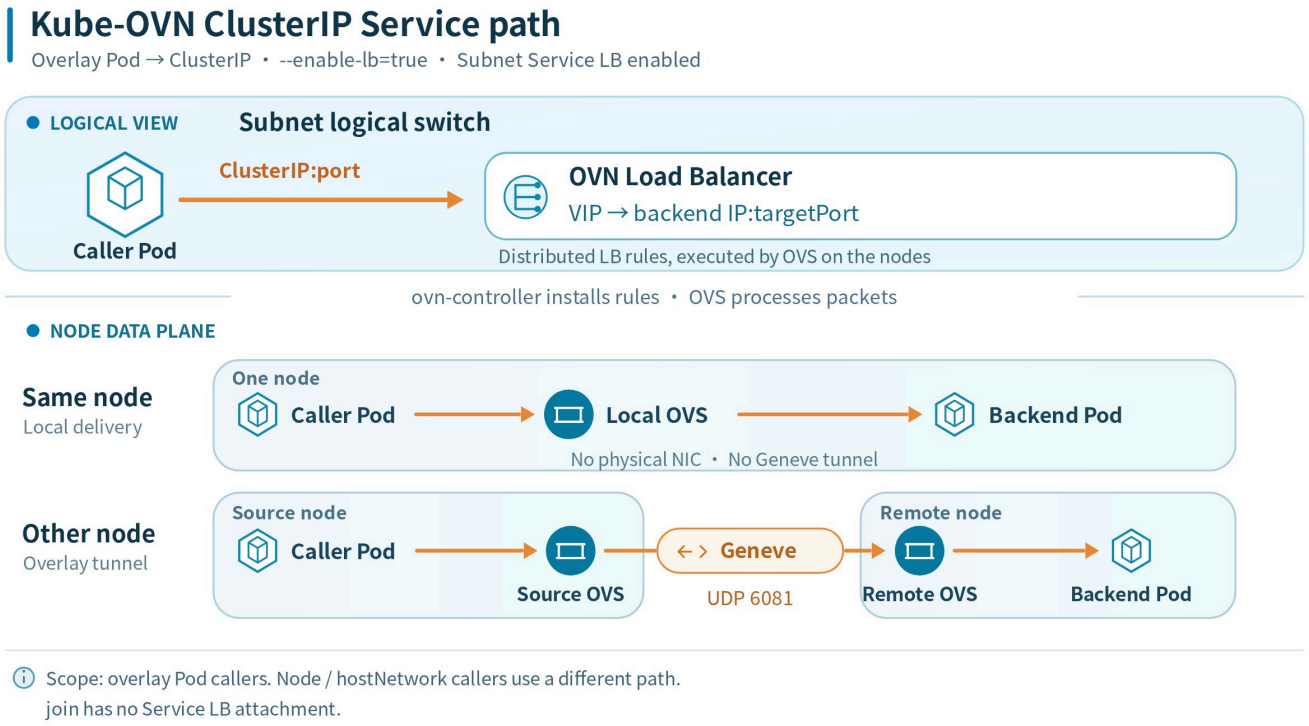
For Pods in the same Subnet on different nodes, the logical switch connects their logical ports, which are bound to different node chassis. The source OVS encapsulates the packet in Geneve, sends it through the node network, and the destination OVS decapsulates it and delivers it to the destination Pod.



For Pods in different Subnets in the same VPC, the logical router also routes the packet between the Subnets. The cross-node overlay transport still uses the Geneve tunnel between the OVS instances.

Pod to a Service

When OVN Service load balancing is enabled for the controller and the Subnet, `kube-ovn-controller` maintains OVN load balancers on eligible Subnet logical switches. OVS translates the Service ClusterIP and port to a selected backend endpoint IP and target port. For a Pod backend, forwarding then follows the same-node or cross-node path above.



The `join` switch is excluded from OVN Service load-balancer attachment. The diagram covers ordinary overlay Pod callers with Pod backends. Other Service access scenarios have additional conditions:

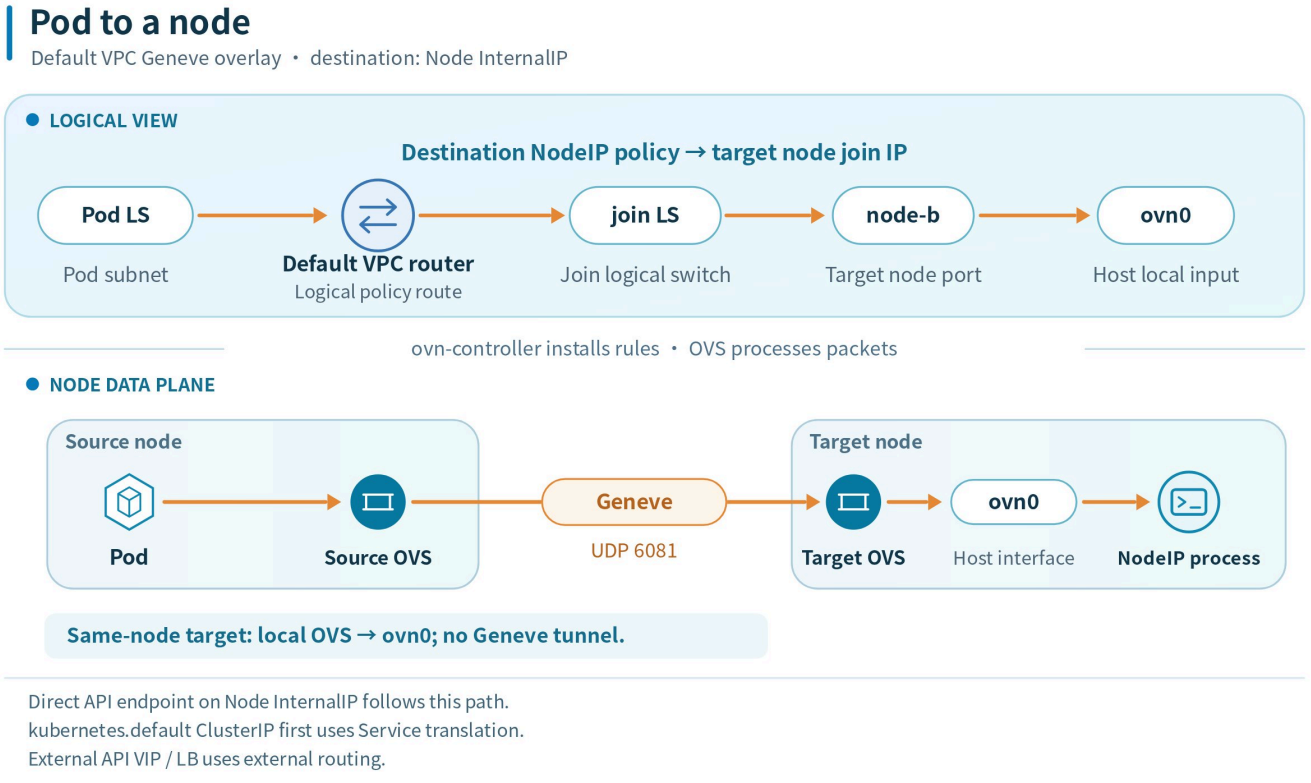
Access scenario	What to consider
Ordinary overlay Pod to ClusterIP	The diagram applies when both the controller and the source Subnet enable OVN Service load balancing. Disabling it changes how Service traffic is handled; verify the cluster's Service implementation with the administrator.
Node or `hostNetwork` Pod to ClusterIP	Traffic originates in the host network and depends on the Service implementation and routing configured on the node. It does not enter through the Pod logical port shown in the diagram.

Access scenario	What to consider
External client to NodePort or LoadBalancer	The configured exposure mechanism determines how traffic enters the cluster. Gateway mode and traffic policies can affect reachability and source IP preservation.
Pod to a headless Service	There is no ClusterIP to translate. With Pod endpoints, clients connect to the endpoint addresses using the Pod-to-Pod paths.

For Service creation and exposure, see Configure Services. Before combining external access with a centralized gateway, check its NodePort and hostport limitations.

Pod to a node or an API server

For traffic to a Node InternalIP, the default VPC logical router uses a destination policy route to direct the packet to that node's address on the `join` switch. The packet reaches the target host through its `ovn0` interface. Traffic to a remote node uses Geneve between the OVS instances; traffic to the local node stays local.



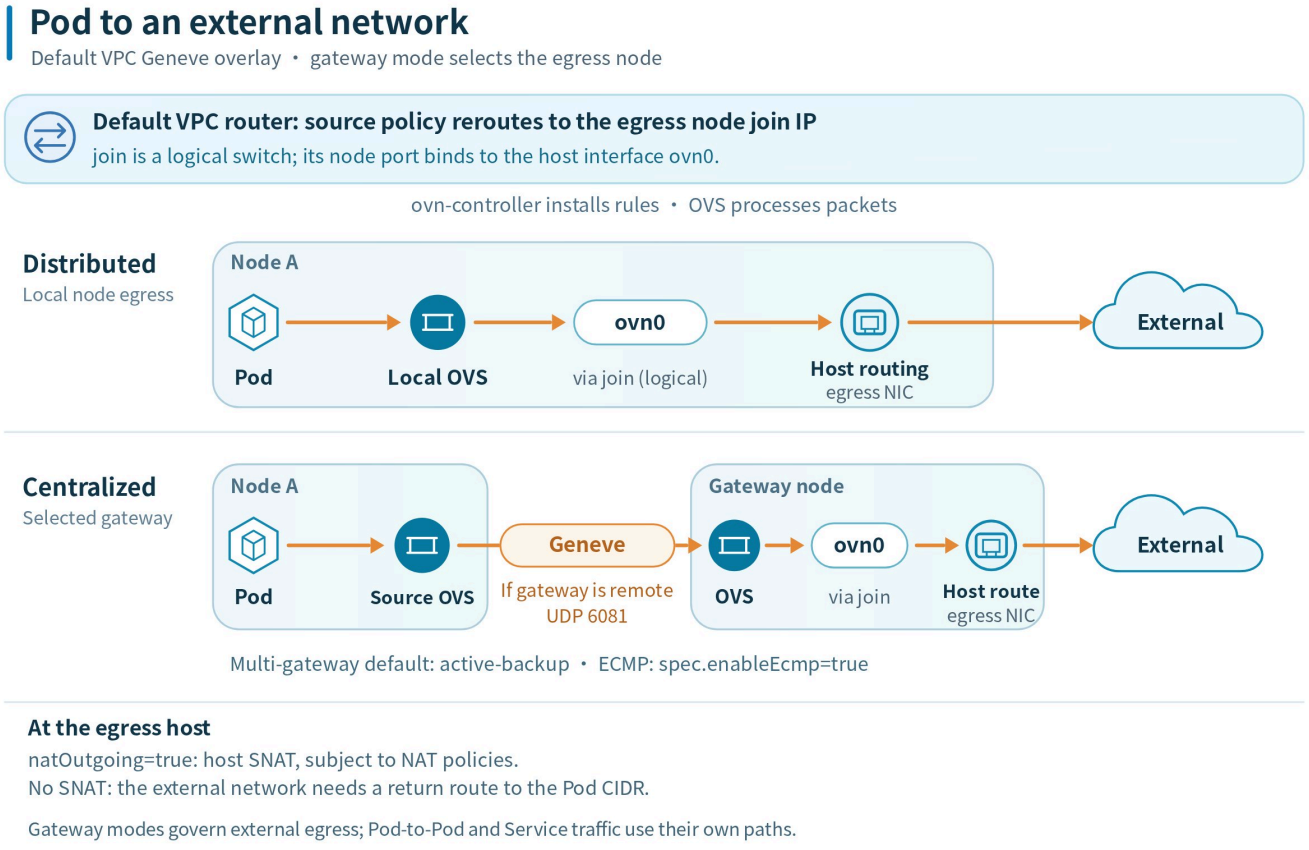
A direct API server endpoint on a Node InternalIP follows this node access path. Access through the `kubernetes` Service ClusterIP first translates the Service address to an API

server endpoint; forwarding then follows the route for that endpoint. An API endpoint exposed through an external VIP or load balancer follows the external routing path instead.

Pod to an external network

The default VPC logical router uses a source policy route to select the egress node's address on the `join` switch. The packet reaches that node through `ovn0`, then follows the host's routing rules and egress NIC to the external network:

- **Distributed gateway**: traffic leaves through the Pod's local node.
- **Centralized gateway**: traffic leaves through a selected gateway node. When that gateway is remote, OVS transports the packet to it over Geneve. See Gateway modes for active-backup and ECMP behavior.



With `natOutgoing: true`, matching external traffic is source-NATed on the egress host, subject to configured NAT policies. If packets leave the egress host with their original Pod source addresses, the upstream network needs a return route to the Pod CIDR. See Configure a centralized gateway for gateway configuration. For workload-specific egress through dedicated gateway Pods, see Egress Gateway and its comparison with centralized gateways.

High Availability

In a high-availability deployment, `ovn-central` replicates the northbound and southbound databases with Raft. Each database needs a reachable majority of its members to commit updates. Database availability is one part of network availability: the controllers, node networking, and any selected gateway must also work.

Failure	Effect on workloads
`kube-ovn-controller` is unavailable	Existing forwarding rules can remain in use, but changes such as new Pod networks, Service endpoints, or policies cannot be reconciled until a controller is available.
OVN databases are unavailable	Already installed rules can continue forwarding while the node data plane and the required paths remain intact. New or changed network configuration cannot be fully applied.
Node OVS components restart, or a node fails	Traffic through that node can be interrupted, including traffic for its Pods and any gateway role it serves. Database replication alone does not prevent this interruption.
A centralized gateway node fails	External connectivity depends on the configured gateway redundancy and failover mode. Connections can be interrupted during failover; see Centralized gateway behavior.

When `ovn-controller` must reload network state after a restart, it needs access to the southbound database and then programs the local OVS. `ovs-vswitchd` does not read the southbound database directly. Connectivity can be interrupted while the node's interfaces and forwarding rules recover.

Inspecting Components in Your Cluster

Use `kubectl` configured for the target cluster and an account with permission to read these resources. The following example outputs were captured from a three-node test cluster running Kube-OVN `v1.16.13`. Replica counts, Pod names, node names, IP addresses, ages, and events vary by cluster; use them to understand the output rather than as required configuration values.

Long outputs are shortened to the key fields and representative rows discussed below. In excerpts, `...` marks omitted lines; running the commands returns the full output.

List workloads

```
kubectl get deployments -n kube-system \
  ovn-central kube-ovn-controller kube-ovn-monitor
```

Example output:

```
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
ovn-central         3/3     3            3           52m
kube-ovn-controller 3/3     3            3           52m
kube-ovn-monitor    1/1     1            1           52m
```

```
kubectl get daemonsets -n kube-system \
  ovs-ovn kube-ovn-cni kube-ovn-pinger
```

Example output:

```
NAME              DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NO
DE SELECTOR            AGE
ovs-ovn           3         3         3       3            3           ku
bernetes.io/os=linux   52m
kube-ovn-cni      3         3         3       3            3           ku
bernetes.io/os=linux   52m
kube-ovn-pinger   3         3         3       3            3           ku
bernetes.io/os=linux   52m
```

Check the Deployment `READY` column against its requested replica count. For each DaemonSet, compare `DESIRED`, `CURRENT`, and `READY`; after a successful rollout, the ready count should match the desired count for eligible nodes. In this example, `ovn-central` and `kube-ovn-controller` each have three ready replicas, `kube-ovn-monitor` has one, and each DaemonSet has a ready Pod on all three eligible nodes.

Locate Pods and containers

List the Pods to see their actual placement and runtime status:

```
kubectl get pods -n kube-system \
  -l 'app in (ovn-central,kube-ovn-controller,ovs,kube-ovn-cni,kube-ovn-p
inger,kube-ovn-monitor,kube-ovn-speaker)' \
  -o wide
```

Example output (excerpt; one Pod per component is shown):

```
NAME                                    READY   STATUS    RESTARTS   AGE
IP           NODE                     NOMINATED NODE   READINESS GATES
kube-ovn-cni-7ctr7                      1/1     Running   0          52m
10.64.0.33   ctyun-k8s3-20261014-c    <none>           <none>
...
kube-ovn-controller-78b79f69bf-6lcn4    1/1     Running   0          52m
10.64.0.73   ctyun-k8s3-20261014-a    <none>           <none>
...
kube-ovn-monitor-56d9f6cd47-47q5k       1/1     Running   0          52m
10.64.0.5    ctyun-k8s3-20261014-b    <none>           <none>
kube-ovn-pinger-j77zk                   1/1     Running   0          52m
10.16.0.5    ctyun-k8s3-20261014-b    <none>           <none>
...
ovn-central-6c784885b4-2flx8            1/1     Running   0          52m
10.64.0.73   ctyun-k8s3-20261014-a    <none>           <none>
...
ovs-ovn-bbgjn                           1/1     Running   0          52m
10.64.0.33   ctyun-k8s3-20261014-c    <none>           <none>
...
```

The `NODE` column shows where each Pod runs. `READY`, `STATUS`, and `RESTARTS` help identify Pods that have not become ready or are restarting. The Pod selector for `ovs-ovn` is `app=ovs`; the other components in this list use their component name as the `app` value. An optional component that is not installed will have no matching Pods; `kube-ovn-speaker` is not installed in this example cluster.

For example, list the containers inside the OVN and OVS Pods:

```
kubectl get pods -n kube-system -l 'app in (ovn-central,ovs)' \
  -o 'custom-columns=POD:.metadata.name,NODE:.spec.nodeName,CONTAINERS:.s
pec.containers[*].name'
```

Example output (excerpt; one Pod per workload is shown):

```
POD                              NODE                     CONTAINERS
ovn-central-6c784885b4-2flx8     ctyun-k8s3-20261014-a    ovn-central
...
ovs-ovn-bbgjn                    ctyun-k8s3-20261014-c    openvswitch
...
```

For the standard deployment, the `CONTAINERS` column shows `ovn-central` for the database Pods and `openvswitch` for the OVS Pods. NB, SB, and `ovn-northd` run as processes inside the `ovn-central` container; the node OVS database, `ovs-vswitchd`, and `ovn-controller` run inside `openvswitch`. They are not separate Kubernetes Pods or containers.

Inspect rollout state and events

To inspect rollout details and workload events, describe the workload. For example:

```
kubectl describe deployment -n kube-system ovn-central
```

Example output (excerpt):

```
Name:                   ovn-central
Namespace:              kube-system
...
Replicas:               3 desired | 3 updated | 3 total | 3 available | 0
unavailable
...
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
  Progressing    True    NewReplicaSetAvailable
...
```

In this example, all three replicas are available, and the `Available` and `Progressing` conditions are `True`.

```
kubectl describe daemonset -n kube-system kube-ovn-cni
```

Example output (excerpt; repeated creation events are omitted):

```
Name:           kube-ovn-cni
Namespace:      kube-system
...
Desired Number of Nodes Scheduled: 3
Current Number of Nodes Scheduled: 3
...
Number of Nodes Scheduled with Available Pods: 3
Number of Nodes Misscheduled: 0
Pods Status:  3 Running / 0 Waiting / 0 Succeeded / 0 Failed
...
Events:
  Type    Reason            Age   From                  Message
  ----    ------            ----  ----                  -------
  Normal  SuccessfulCreate  52m   daemonset-controller  Created pod: kube
-ovn-cni-7ctr7
...
```

Here, the desired, current, and available node counts are all three, and no Pods are waiting or failed. The `SuccessfulCreate` event shown records creation of one Pod. Events are

transient, so an older workload can show `Events: <none>`; check current readiness as well as any events.

These queries show deployment and readiness state. Use the traffic paths and the linked configuration guides to check the behavior of a particular network connection.

Resource usage and limits

Insufficient memory can cause the `openvswitch` container to be `OOMKilled`; CPU throttling can delay network configuration and packet processing. Ask the cluster administrator to check resource usage when these symptoms occur.

The ACP Helm Chart uses `ovs-ovn.requests` and `ovs-ovn.limits` for resource settings. Its upgrade template reads CPU and memory values from the existing `ovs-ovn` DaemonSet, so administrators must also verify the effective resources on that DaemonSet when changing or upgrading the configuration.

kubectl-ko

This kubectl plugin lets cluster administrators inspect OVN databases and node OVS state. It wraps commands such as `ovn-nbctl`, `ovn-sbctl`, `ovs-vsctl`, and `ovs-ofctl`. Use the plugin version supplied for the installed Kube-OVN release and the permissions required to access its components.

Next Steps

Task	Guide
Select a Pod network and plan its address range	Configure Subnets and Configure IPPool
Control which workloads can communicate	Network Policy
Limit Pod ingress and egress bandwidth (QoS)	Configure Pod network bandwidth limits

Task	Guide
Provide a stable application access address	Configure Services
Select an outbound gateway	Configure Centralized Gateway and Configure Egress Gateway
Connect Pods to the physical network	Prepare the underlay physical network and Configure underlay Subnets
Connect underlay and overlay Subnets	Automatic underlay and overlay interconnection
Add another network interface to a Pod	Pod multi-network interfaces
Account for tunnel overhead	Configure MTU

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

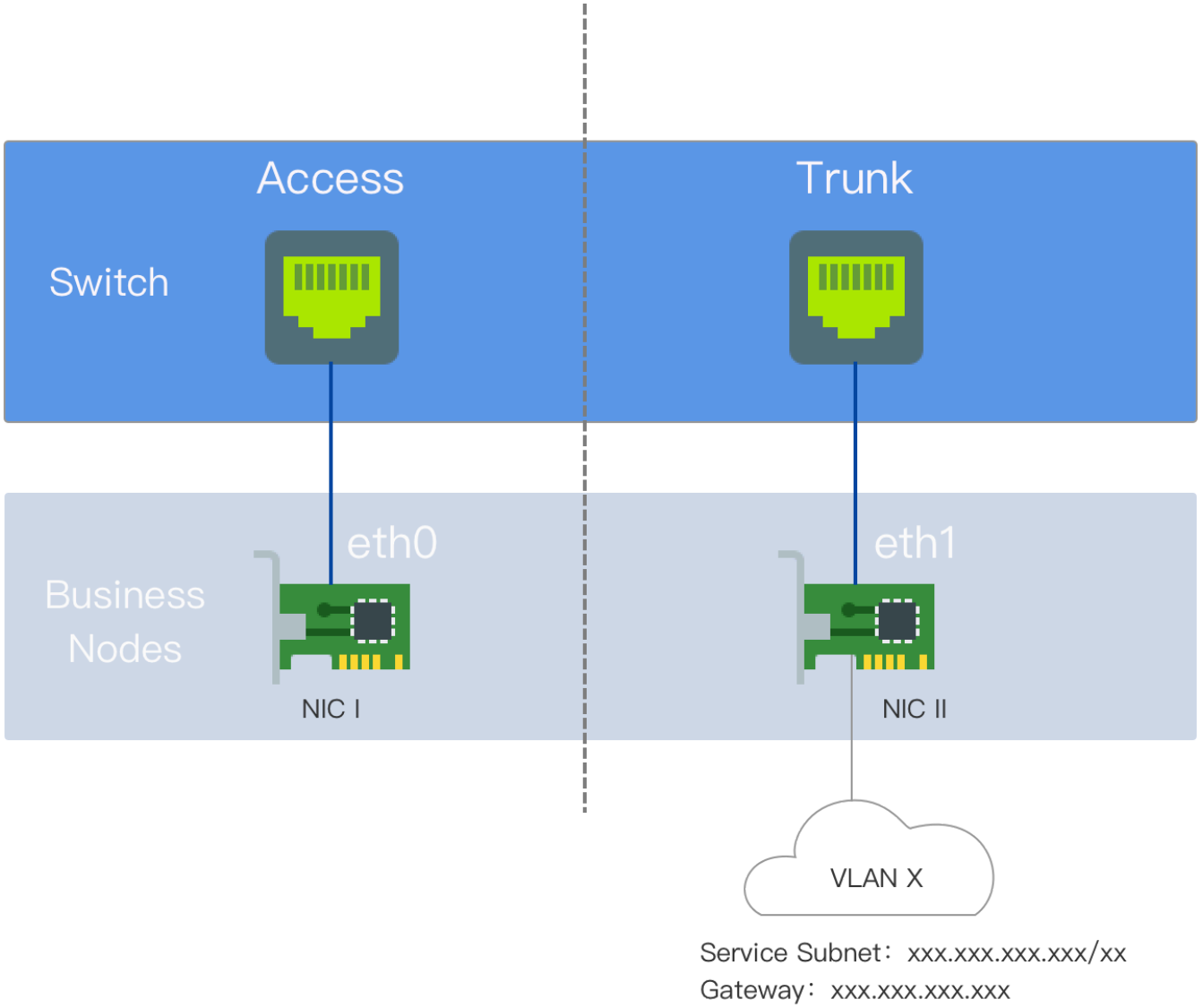
Usage Instructions

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.
  - If workloads require IPv4 multicast communication, the physical network must permit the required IPv4 multicast groups and IGMP traffic. For configuration details, see Configure IPv4 Multicast for Kube-OVN Underlay Networks.

- **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 thro
ugh
#
```

Check Network Connectivity

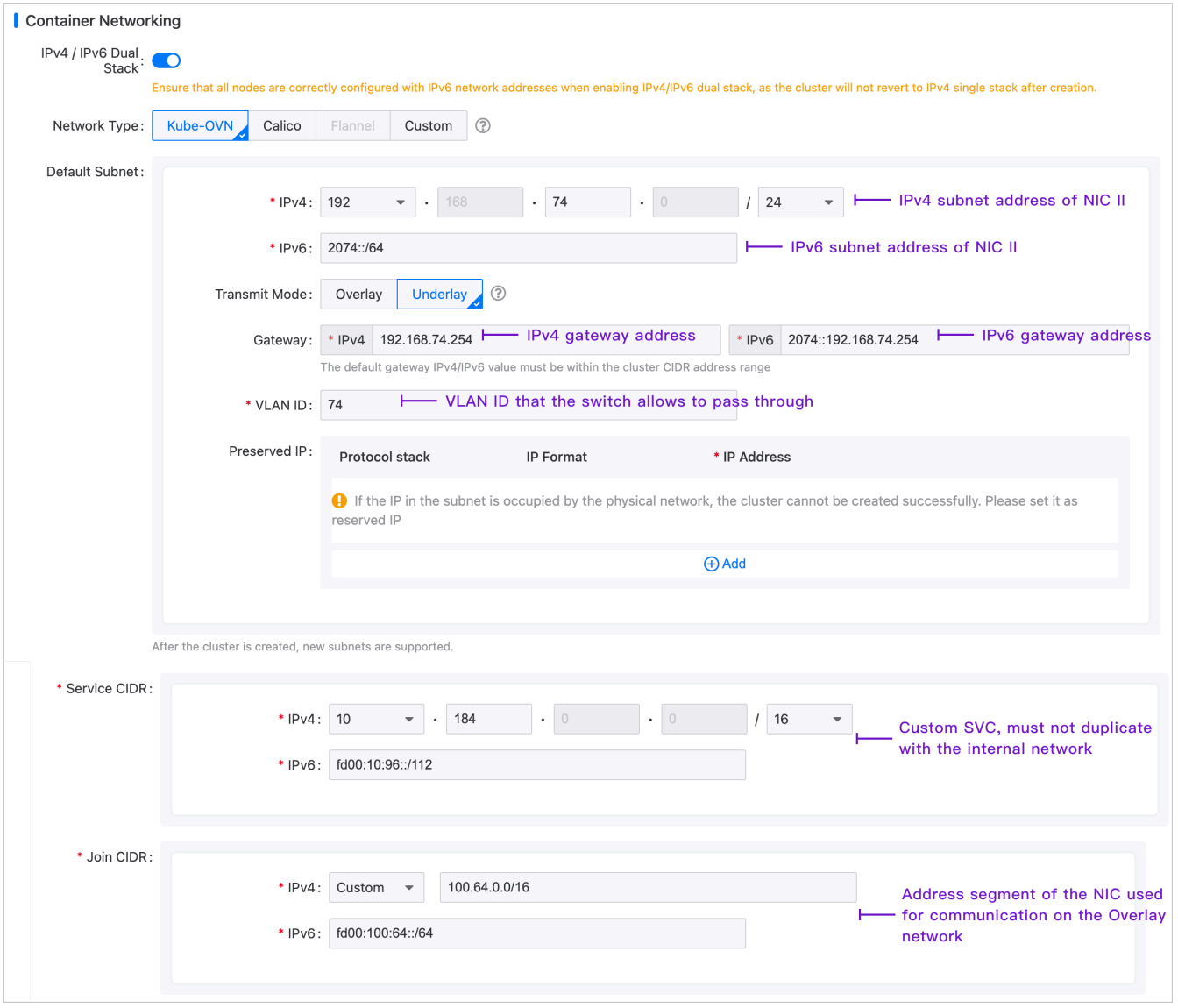
Test if NIC Two can communicate with the gateway address:

```
ip link add ens224.74 link ens224 type vlan id 74  // The NIC name is ens
224, 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 wit
hin 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 a
fter 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 cluster creation path selection, see Clusters Overview, 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.



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

Optional: Restrict U2O Routing to Overlay CIDRs

Isolation Between Underlay Subnets with u2oInterconnection Enabled

Step 1: Configure kube-ovn-controller

Step 2: Configure Subnet Isolation

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.

Optional: Restrict U2O Routing to Overlay CIDRs

To route only traffic destined for Overlay subnet CIDRs through the U2O gateway, enable `u2oFeatures.overlayOnlyRouting` on the same Underlay subnet. Traffic to other destinations continues to use the physical gateway.

```
spec:
  u2oInterconnection: true
  u2oFeatures:
    overlayOnlyRouting: true
```

This option requires `u2oInterconnection: true`.

Isolation Between Underlay Subnets with u2oInterconnection Enabled

When multiple Underlay subnets have `u2oInterconnection: true` enabled, traffic between them no longer goes through the physical gateway but is routed directly via the internal OVN network.

If you need to isolate two Underlay subnets while both have `u2oInterconnection` enabled, you must first configure the kube-ovn-controller parameter, then configure the subnet isolation.

Step 1: Configure kube-ovn-controller

Modify the kube-ovn-controller Deployment to disable connection tracking skip for destination logical port IPs:

```
kubectl edit deployment kube-ovn-controller -n kube-system
```

Add or modify the following argument:

```
spec:
  template:
    spec:
      containers:
      - name: kube-ovn-controller
        args:
        - --ls-ct-skip-dst-lport-ips=false
```

CAUTION

`--ls-ct-skip-dst-lport-ips` controls whether to skip connection tracking (conntrack) for traffic destined to logical port IPs. The default value is `true`, which skips conntrack to improve performance. Setting it to `false` does not affect functionality but may slightly impact performance.

However, for Underlay subnets with ACL-based isolation, you **must** set it to `false`. Otherwise, gateway-to-Pod traffic will fail (e.g., ping requests reach the Pod but replies are dropped), because ACL isolation uses `allow-related` which requires conntrack state; without it, replies cannot be identified as "related" and get dropped.

Step 2: Configure Subnet Isolation

Configure the subnet with the following parameters:

```yaml
spec:
  u2oInterconnection: true
  acls:
  - action: drop
    direction: to-lport  # Ingress direction (traffic entering the logica
l port)
    match: ip4.src == 172.20.0.0/16
    priority: 1002
  - action: drop
    direction: to-lport  # Ingress direction
    match: ip4.src == 192.50.0.0/16
    priority: 1002
```

ACL Parameters:

Parameter	Description
`action`	The action to take: `allow`, `drop`, or `allow-related`
`direction`	Traffic direction: `to-lport` (ingress) or `from-lport` (egress)
`match`	OVN match expression using L2-L4 fields and boolean operators
`priority`	Rule priority (higher values are evaluated first; recommended range: 1002-1899)

NOTE

- The `acls` field provides priority-based rule evaluation, offering more flexibility than standard Kubernetes NetworkPolicy.
- When using `to-lport` direction, `ip4.src` refers to the source IP of incoming traffic.
- **Recommended priority range**: `1002` to `1899` to avoid conflicts with system default ACL rules.

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
Trademark Notice

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.

> ⚠️ **Critical**: The LoadBalancer VIP and the backend Pod IPs **must be in the same Underlay subnet**.

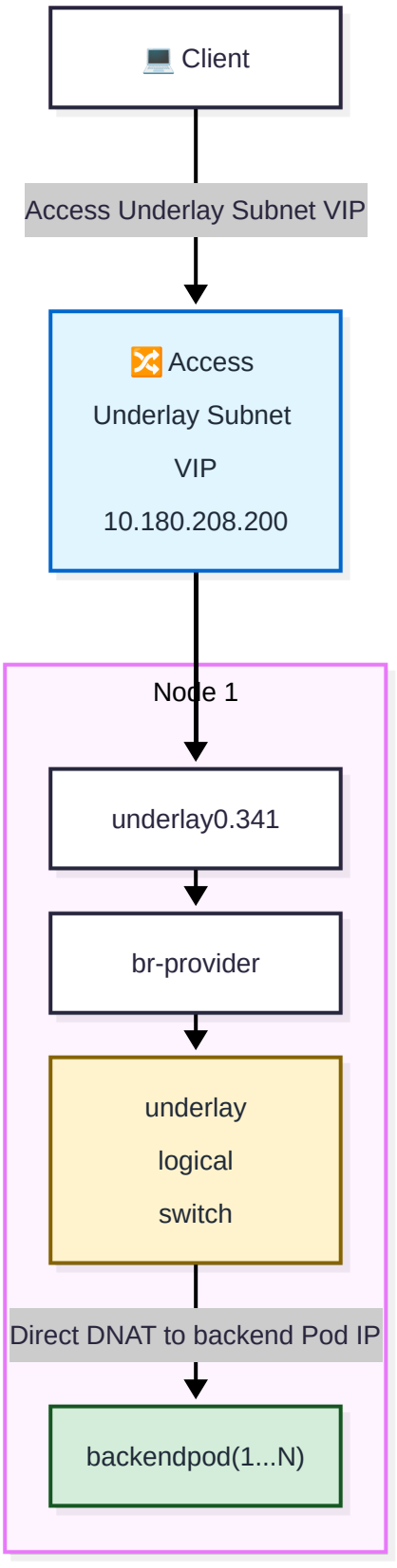
Prerequisites

Environment Requirements

- **ACP version**: >= 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:

```yaml
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
  # If customInterfaces are used, each interface MUST also be the VLAN su
b-interface
  # (parent interface name + VLAN suffix, e.g., underlay0.341), not the p
arent interface
  # customInterfaces:
  #   - interface: underlay0.341
  #     nodes:
  #       - node1

---
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
```

> ⚠️ **Important**: If the `ProviderNetwork` specifies per-node interfaces via `customInterfaces`, each `customInterfaces[].interface` **must also use the VLAN sub-interface** (parent interface name + VLAN suffix, e.g., `underlay0.341`), not the parent

> interface (e.g., `underlay0`). Leaving a parent interface in `customInterfaces` prevents MetalLB LoadBalancer traffic from reaching the backend Pods.

> ⚠️ **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**
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`:

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

4. Create MetalLB External Address Pool

```yaml
# 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
  interfaces:
    - br-provider  # Optional: Interface for ARP broadcasting; use bridge
interface (br-*) instead of physical interface
  nodeSelectors: []
```

Deploy the address pool:

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

5. Create Sample Application and LoadBalancer Service

```yaml
# 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:

```yaml
apiVersion: v1
kind: Service
metadata:
  name: backend-lb-service
  annotations:
    metallb.io/address-pool: acp-underlay-pool  # Use the Underlay addres
s 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
```

Trademark Notice

NGINX® is a registered trademark of F5, Inc. in the U.S. and in other countries.

MetalLB™ is a trademark of The Linux Foundation.

Alauda is an independent vendor. This product is not affiliated with, endorsed by, or sponsored by The Linux Foundation or F5, Inc. All trademarks are the property of their respective owners and are used here for identification purposes only.

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 a control plane node of the cluster to obtain the `install-ic-server.sh` installation script from the kube-ovn-controller Pod.

```
kubectl -n kube-system cp $(kubectl get pods -n kube-system -l app=kube
-ovn-controller -o custom-columns=NAME:.metadata.name --no-headers | he
ad -1):/kube-ovn/install-ic-server.sh ./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: REGIST
RY="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=rbin
d:rw" \
--mount="type=bind,src=/var/log/ovn,dst=/var/log/ovn,options=rbin
d:rw" \
--env="NODE_IPS=<IP addresses of all nodes, separated by commas>"
\   ## For example: --env="NODE_IPS="192.168.178.97,192.168.181.9
3,192.168.177.192""
--env="LOCAL_IP=<IP address of the current node>" \   ## For examp
le: --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 b
ash 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=rbin
d:rw" \
--mount="type=bind,src=/var/log/ovn,dst=/var/log/ovn,options=rbin
d:rw" \
--env="NODE_IPS=<IP addresses of all nodes, separated by commas>"
\   ## For example: --env="NODE_IPS="192.168.178.97,192.168.181.9
3,192.168.177.192"" \
--env="LOCAL_IP=<IP address of the current node>" \   ## For examp
le: --env="LOCAL_IP="192.168.181.93""
--env="LEADER_IP=<IP address of the Leader node>" \   ## For examp
le: --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 b
ash start-ic-db.sh
```

Deploy the cluster interconnection controller in the Global cluster

In any control plane 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 case, the local IP of the three nodes where the controller is deployed
  ic-nb-port: "6645"            # Cluster Interconnect Controller nb port, default 6645
  ic-sb-port: "6646"            # Cluster Interconnect Controller sb port, 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 **workload 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 workload 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

- Overview
- Egress Gateway vs. Centralized Gateway
- How Egress Gateway Works
- Before You Begin
- Configuration Workflow
  - Step 1: Prepare the External Network Attachment
  - Step 2: Create a VPC Egress Gateway
  - Step 3: Validate the Gateway
    1. Check the resource status
    2. Inspect networking inside the gateway Pod
    3. Confirm traffic forwarding
    4. Confirm OVN routing policies
  - Optional: Enable Multi-Replica Load Balancing
  - Optional: Configure Bandwidth Limits
  - Optional: Enable BFD-based High Availability
    1. Enable a BFD Port on the VPC
    2. Enable BFD on the VPC Egress Gateway
    3. Verify BFD status
- Validated Capacity Guidance
- Operations That May Interrupt Traffic

Additional Resources

Overview

Egress Gateway, also called VPC Egress Gateway, provides stable outbound addresses for Pods in an overlay network. It routes selected workloads through dedicated gateway Pods before the traffic leaves the cluster.

Use Egress Gateway when you need:

- Stable source IP addresses for specific workloads
- Workload-level egress control instead of subnet-level control
- Higher throughput through horizontal scaling
- Faster failover for outbound traffic

Main capabilities:

- Active-Active high availability through ECMP, with horizontal throughput scaling
- Fast failover through BFD, typically in less than 1 second
- Support for IPv4, IPv6, and dual-stack environments
- Fine-grained traffic matching through NamespaceSelector and PodSelector
- Flexible scheduling through node selectors and tolerations

Current limitations:

- Multi-replica deployments require multiple egress IPs
- Source NAT mapping records are not retained

Egress Gateway vs. Centralized Gateway

Compare Egress Gateway with Centralized Gateway to choose the gateway model that best fits your scenario. For centralized mode details, refer to Configure Centralized Gateway.

Dimension	Egress Gateway	Centralized Gateway
Granularity	Fine-grained control by `selectors` and `policies` (Namespace/Pod/Subnet/IPBlock).	Applied at subnet level (`gatewayType: centralized`).
Egress Address Source	Uses dedicated gateway Pods with external subnet IPs.	Uses designated gateway nodes; with `natOutgoing: true`, egress uses node IPs.
Data Path	Traffic is routed to VPC Egress Gateway Pods, then SNATed to the external network.	Traffic is routed to designated nodes (`ovn0`) and then forwarded by host routing/NAT rules.
HA and Failover	Active-Active ECMP; supports BFD fast failover (sub-second).	Supports ECMP and primary-backup; failover is generally slower than BFD-based Egress Gateway.
Scheduling Control	Supports node scheduling controls for gateway workloads (`nodeSelector`, `tolerations`).	Gateway nodes are selected by `gatewayNode` or `gatewayNodeSelectors`.
Typical Use Cases	Tenant-level or workload-level egress isolation, fine-grained policy control, higher performance and faster failover.	Simpler subnet-level fixed-source egress, auditing, IP allowlists, and firewall source-IP management.

Selection guidance:

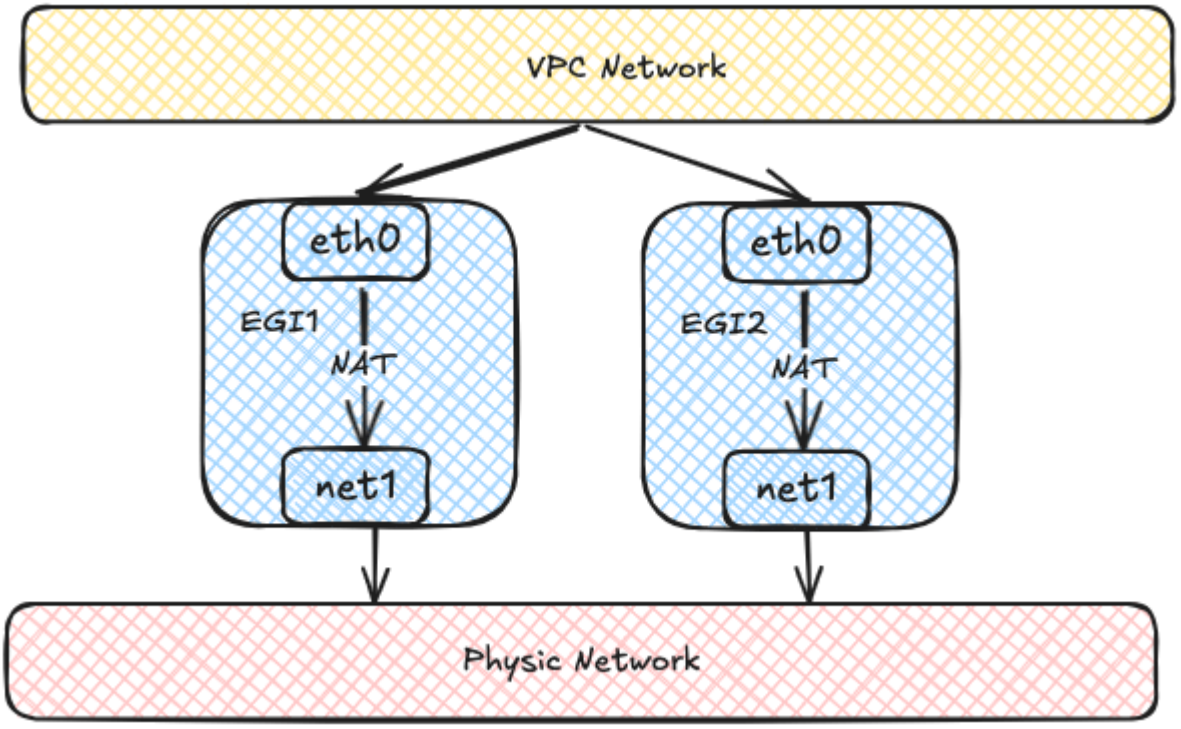
- Choose **Egress Gateway** when you need workload-level policy control, scalable throughput, and fast failover.
- Choose **Centralized Gateway** when subnet-level fixed egress and simple operations are sufficient.

How Egress Gateway Works

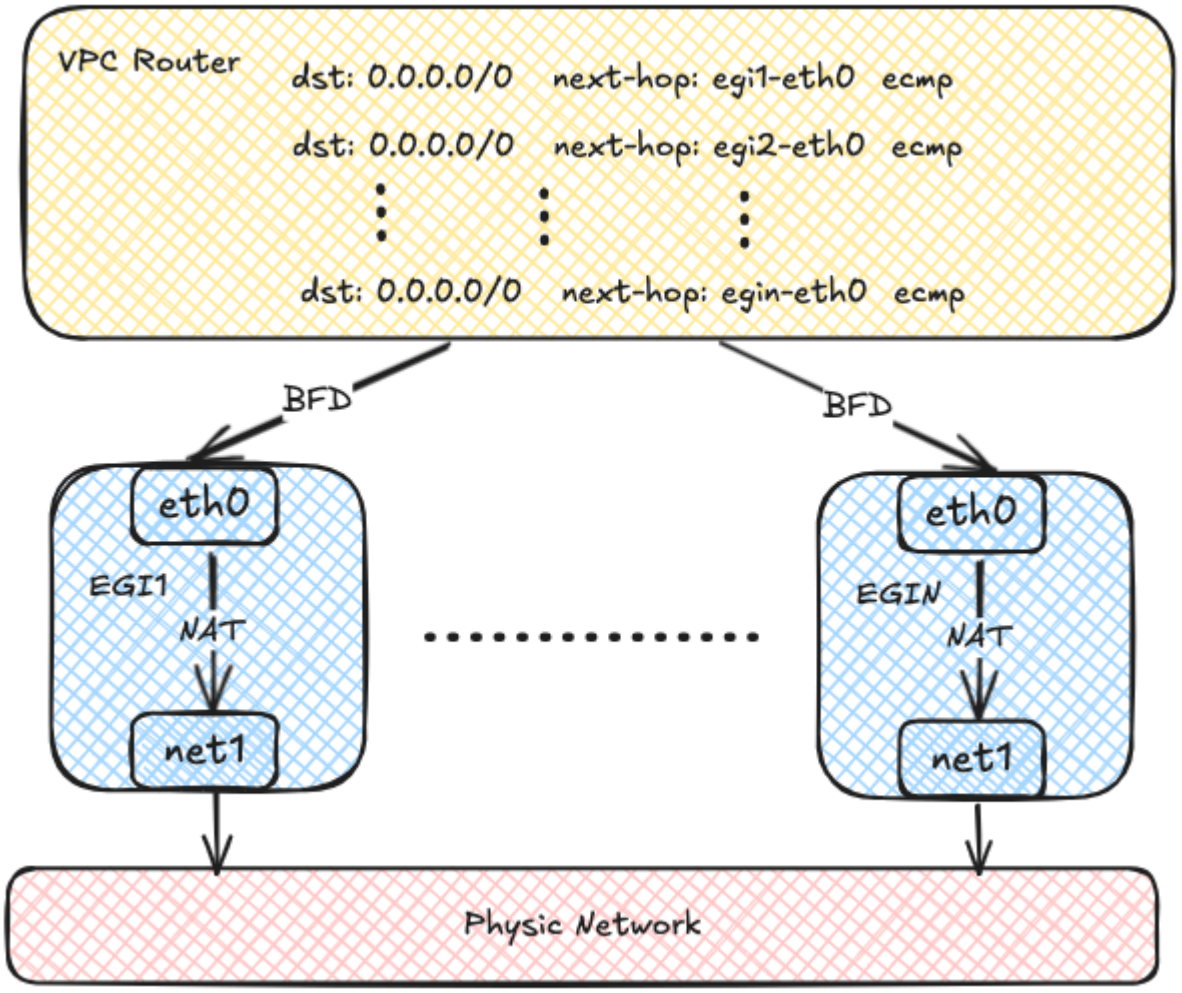
An Egress Gateway runs as one or more Pods. Each Pod uses two network interfaces:

- One interface connects to the overlay network inside the cluster.
- The other interface connects to the external underlay network.

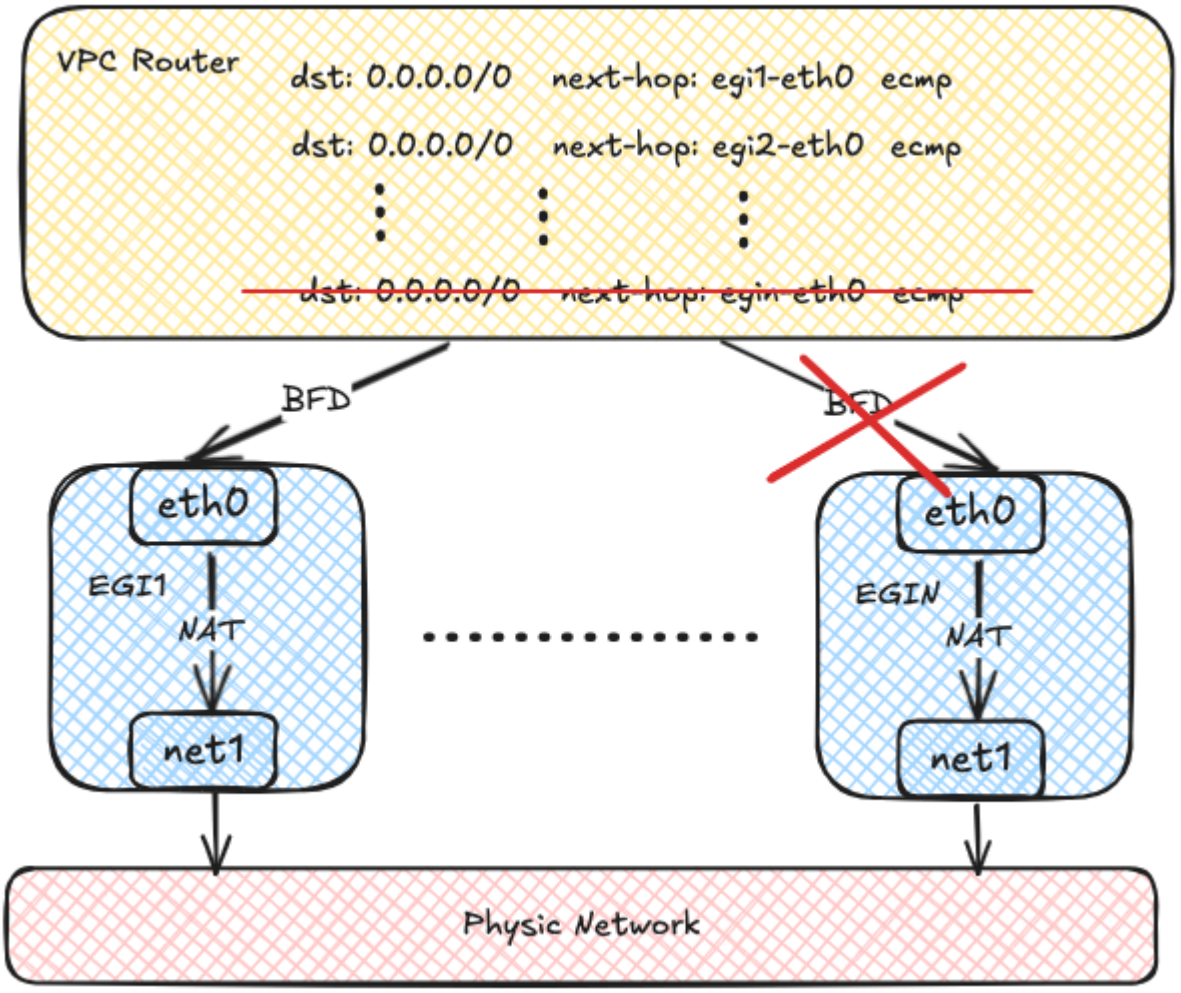
Traffic from selected Pods is first forwarded to the gateway Pods and then sent to the external network through the underlay interface.



Each Egress Gateway instance registers its address in the OVN routing table. When a Pod in the overlay network accesses the external network, OVN uses source-address hashing to distribute traffic across multiple gateway instances. This provides load balancing and lets throughput scale horizontally as more instances are added.



OVN can use BFD to probe multiple gateway instances. If one instance fails, OVN marks the corresponding route as unavailable and quickly redirects traffic to a healthy instance.



Before You Begin

Before you start, make sure the following prerequisites are met:

- *Multus CNI Plugin* **MUST** already be installed on the cluster.
- The external underlay network, VLAN, and bridge network are already planned and available.
- You have identified which workloads should use the gateway and whether they require SNAT.

To install *Multus CNI Plugin*, refer to Deploying the Multus CNI Plugin.

Configuration Workflow

Configure Egress Gateway in the following order:

1. Prepare the external network attachment, including the subnet and Network Attachment Definition.
2. Create a VPC Egress Gateway resource and specify the external subnet and policies.
3. Verify resource status, routes, and traffic forwarding.
4. Optionally scale out replicas and enable BFD-based fast failover.

Step 1: Prepare the External Network Attachment

Egress Gateway uses multiple interfaces to connect to both the internal and external networks. Before creating the gateway, prepare the following resources:

- An external subnet
- A Network Attachment Definition (NAD) for that subnet

> **NOTE**
>
> The NetworkAttachmentDefinition name MUST NOT contain a dot (`.`). Each NetworkAttachmentDefinition can be used by only one subnet. Do not reuse the same NetworkAttachmentDefinition across multiple subnets.

The following example uses a Kube-OVN underlay subnet as the external network.

> **NOTE**
>
> This example assumes that the bridge network and VLAN resource `external-vlan` have already been created for the underlay network.

```yaml
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  name: underlay-ext
  namespace: default
spec:
  config: |-
    {
      "cniVersion": "0.3.0", (1)
      "type": "kube-ovn", (2)
      "server_socket": "/run/openvswitch/kube-ovn-daemon.sock", (3)
      "provider": "underlay-ext.default.ovn"
    }
---
apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
  name: underlay-ext
spec:
  protocol: IPv4
  provider: underlay-ext.default.ovn (4)
  cidrBlock: 172.17.0.0/16 (5)
  gateway: 172.17.0.1 (6)
  vlan: external-vlan (7)
  excludeIps: (8)
    - 172.17.0.11..172.17.0.20
```

(1) Use the Kube-OVN CNI plugin for the secondary network.

(2) Kube-OVN daemon socket used by the CNI plugin.

(3) Provider name in the format `<network attachment definition name>.<namespace>.ovn`.

(4) The provider used by the subnet. This value MUST match the provider in the NetworkAttachmentDefinition.

(5) CIDR of the external underlay network.

(6) Gateway of the external underlay network.

(7) VLAN resource used by the underlay subnet.

(8) IP range excluded from automatic allocation. For details, refer to Example Subnet custom resource (CR) with Kube-OVN Underlay Network.

TIP

Before using an underlay subnet, make sure the physical network, VLAN, and bridge network are prepared correctly. For environment planning details, refer to Preparing Kube-OVN Underlay Physical Network.

Step 2: Create a VPC Egress Gateway

Create a VPC Egress Gateway resource and define which workloads should use it.

```yaml
apiVersion: kubeovn.io/v1
kind: VpcEgressGateway
metadata:
  name: gateway1
  namespace: default (1)
spec:
  replicas: 1 (2)
  internalSubnet: ovn-default (3)
  externalSubnet: underlay-ext (4)
  externalIPs: (5)
    - 172.17.0.11
    - 172.17.0.12
  resources: (6)
    requests:
      cpu: 100m
      memory: 128Mi
    limits:
      cpu: 200m
      memory: 256Mi
      ephemeral-storage: 2Gi
  nodeSelector: (7)
    - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
            - node1
            - node2
  tolerations: (8)
    - key: node-role.kubernetes.io/control-plane
      operator: Exists
      effect: NoSchedule
  selectors: (9)
    - namespaceSelector:
        matchLabels:
          kubernetes.io/metadata.name: ns1
    - namespaceSelector:
        matchLabels:
          kubernetes.io/metadata.name: ns2
      podSelector:
        matchLabels:
          app: myapp
  policies: (10)
    - snat: true (11)
```

```
      subnets: 12
        - subnet1
    - snat: false
      ipBlocks: 13
        - 10.18.0.0/16
```

1 Namespace where the VPC Egress Gateway instances are created.

2 Number of VPC Egress Gateway instances.

3 Internal subnet that connects to the internal network. The subnet MUST be an overlay subnet in the same VPC and have enough free IPs for the gateway instances. If not specified, the gateway Pods will use the default internal subnet of the VPC.

4 External subnet that connects to the external network.

5 External IPs used by the gateway Pods on the underlay network. Each gateway instance is allocated one IP from this list. These IPs MUST be within the CIDR of the external subnet and should be included in the `excludeIps` range of the subnet. It's recommended to reserve *.spec.replicas + 1* IPs so that a gateway Pod can still obtain an IP in edge cases.

6 Resource requests and limits for each VPC Egress Gateway instance. If not specified, the default resource requests and limits defined in the VPC Egress Gateway controller will be applied.

7 Node selectors used for scheduling the VPC Egress Gateway instances.

8 Tolerations used for scheduling the VPC Egress Gateway instances.

9 Namespace selectors and Pod selectors used to select Pods that access the external network via the VPC Egress Gateway.

10 Policies for the VPC Egress Gateway, including SNAT and subnets/ipBlocks to be applied.

11 Whether to enable SNAT for the policy.

12 Subnets to which the policy applies.

13 IP blocks to which the policy applies.

This example creates a VPC Egress Gateway named *gateway1* in the `default` namespace. Traffic that matches the selectors and policies is forwarded through the external subnet *underlay-ext*. In this example, that includes:

- Pods in the *ns1* namespace
- Pods in the *ns2* namespace with the label `app: myapp`

- Traffic related to the *subnet1* subnet
- Traffic related to the CIDR *10.18.0.0/16*

> **NOTE**
>
> Pods matching *.spec.selectors* are always SNATed by the gateway.

Step 3: Validate the Gateway

After creating the gateway, confirm that it is ready and forwarding traffic as expected.

1. Check the resource status

Start with the basic resource status:

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

Then check the detailed gateway information:

```
kubectl get veg gateway1 -o wide
NAME       VPC           REPLICAS   BFD ENABLED   EXTERNAL SUBNET   PHASE
READY   INTERNAL IPS     EXTERNAL IPS      WORKING NODES   AGE
gateway1   ovn-cluster   1          false         underlay-ext      Compl
eted   true    ["10.16.0.12"]   ["172.17.0.11"]   ["node1"]       82s
```

Finally, verify that the gateway workloads are running:

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

$ kubectl get pod -n default -l ovn.kubernetes.io/vpc-egress-gateway=gate
way1 -o wide
NAME                        READY   STATUS    RESTARTS   AGE     IP
NODE    NOMINATED NODE   READINESS GATES
gateway1-b9f8b4448-76lhm   1/1     Running   0          4m48s   10.16.0.1
2   node1   <none>           <none>
```

2. Inspect networking inside the gateway Pod

Inspect IP addresses, routing entries, and iptables rules inside the gateway Pod:

```
$ kubectl exec -n default gateway1-b9f8b4448-76lhm -c gateway -- ip addre
ss 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 st
ate 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 s
tate 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 -n default 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 -n default 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 -n default gateway1-b9f8b4448-76lhm -c gateway -- ip route
```

```
show table 1000
default via 10.16.0.1 dev eth0

$ kubectl exec -n default 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
```

3. Confirm traffic forwarding

Capture packets in the gateway Pod to confirm that traffic is forwarded through the gateway:

```
$ kubectl exec -n default gateway1-b9f8b4448-76lhm -c gateway -- 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), snapsh
ot length 262144 bytes
06:50:58.936528 eth0  In  ifindex 17 92:26:b8:9e:f2:1c ethertype IPv4 (0x
0800), length 104: (tos 0x0, ttl 63, id 30481, offset 0, flags [DF], prot
o 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 (0x0
800), 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 (0x0
800), length 104: (tos 0x0, ttl 64, id 26701, offset 0, flags [none], pro
to 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 (0x
0800), length 104: (tos 0x0, ttl 63, id 26701, offset 0, flags [none], pr
oto ICMP (1), length 84)
    172.17.0.1 > 10.17.0.9: ICMP echo reply, id 37989, seq 0, length 64
```

4. Confirm OVN routing policies

OVN Logical Router policies are created automatically for the selected traffic:

```
$ 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  1
00.64.0.4
     30000                              ip4.dst == 172.18.0.3  reroute  1
00.64.0.3
     30000                              ip4.dst == 172.18.0.4  reroute  1
00.64.0.2
     29100                    ip4.src == $VEG.8ca38ae7da18.ipv4  reroute  1
0.16.0.12 1
     29100                     ip4.src == $VEG.8ca38ae7da18_ip4  reroute  1
0.16.0.12 2
     29000 ip4.src == $ovn.default.kube.ovn.control.plane_ip4  reroute  1
00.64.0.3
     29000       ip4.src == $ovn.default.kube.ovn.worker2_ip4  reroute  1
00.64.0.2
     29000        ip4.src == $ovn.default.kube.ovn.worker_ip4  reroute  1
00.64.0.4
     29000     ip4.src == $subnet1.kube.ovn.control.plane_ip4  reroute  1
00.64.0.3
     29000           ip4.src == $subnet1.kube.ovn.worker2_ip4  reroute  1
00.64.0.2
     29000            ip4.src == $subnet1.kube.ovn.worker_ip4  reroute  1
00.64.0.4
```

1 Logical Router Policy used by the VPC Egress Gateway to forward traffic matched by *.spec.policies*.

2 Logical Router Policy used by the VPC Egress Gateway to forward traffic matched by *.spec.selectors*.

Optional: Enable Multi-Replica Load Balancing

> **NOTE**
>
> Before scaling out replicas, make sure to prepare enough external IPs in the external subnet and specify them in *.spec.externalIPs*.

To enable ECMP load balancing and scale throughput horizontally, increase *.spec.replicas*:

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

$ kubectl get veg -n default gateway1
NAME       VPC           REPLICAS   BFD ENABLED   EXTERNAL SUBNET   PHASE
READY   AGE
gateway1   ovn-cluster   2          false         underlay-ext      Compl
eted   true    39m

$ kubectl get pod -n default -l ovn.kubernetes.io/vpc-egress-gateway=gate
way1 -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
node1   <none>           <none>
gateway1-b9f8b4448-zd4dl   1/1     Running   0          64s   10.16.0.13
node2   <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  1
00.64.0.4
     30000                              ip4.dst == 172.18.0.3  reroute  1
00.64.0.3
     30000                              ip4.dst == 172.18.0.4  reroute  1
00.64.0.2
     29100                  ip4.src == $VEG.8ca38ae7da18.ipv4  reroute  1
0.16.0.12, 10.16.0.13
     29100                   ip4.src == $VEG.8ca38ae7da18_ip4  reroute  1
0.16.0.12, 10.16.0.13
     29000 ip4.src == $ovn.default.kube.ovn.control.plane_ip4  reroute  1
00.64.0.3
     29000       ip4.src == $ovn.default.kube.ovn.worker2_ip4  reroute  1
00.64.0.2
     29000        ip4.src == $ovn.default.kube.ovn.worker_ip4  reroute  1
00.64.0.4
     29000     ip4.src == $subnet1.kube.ovn.control.plane_ip4  reroute  1
00.64.0.3
     29000           ip4.src == $subnet1.kube.ovn.worker2_ip4  reroute  1
00.64.0.2
```

```
     29000            ip4.src == $subnet1.kube.ovn.worker_ip4  reroute  1
00.64.0.4
```

Optional: Configure Bandwidth Limits

To limit the bandwidth of selected namespaces or workloads, set *.spec.bandwidth.ingress* and *.spec.bandwidth.egress*. Each field accepts one of the following formats:

- A non-negative integer or numeric string, interpreted as Mbps. For example, `2000` and `"2000"` both mean 2000 Mbps.
- A Kubernetes quantity string with the `M`, `Mi`, `G`, or `Gi` suffix, interpreted as bits per second. Decimal values are supported when a suffix is present. The value is rounded up to the nearest whole Mbps. For example, `1Gi` becomes 1074 Mbps.

Omit a field or set it to `0` to leave that direction unlimited. The limits apply to each gateway instance.

> **NOTE**
>
> If *.spec.replicas* is greater than 1, each replica gets the configured limit. Quantity strings require a VPC Egress Gateway CRD and controller version that supports them. During a rolling upgrade, continue to use integer Mbps values until the Kube-OVN upgrade is complete.

The following example limits the selected namespaces to 2 Gbps ingress bandwidth and 4 Gbps egress bandwidth:

```yaml
apiVersion: kubeovn.io/v1
kind: VpcEgressGateway
metadata:
  name: gateway-bandwidth
  namespace: default
spec:
  replicas: 1
  internalSubnet: ovn-default
  externalSubnet: underlay-ext
  externalIPs:
    - 172.17.0.21
  bandwidth:
    ingress: 2G
    egress: 4G
  selectors:
    - namespaceSelector:
        matchExpressions:
          - key: kubernetes.io/metadata.name
            operator: In
            values:
              - ecommerce
              - ecommerce-test
```

In this example, the direction is relative to the selected namespaces or workloads: `ingress` limits traffic from the external network to them, and `egress` limits traffic from them to the external network.

Optional: Enable BFD-based High Availability

BFD-based failover depends on the VPC BFD LRP. Enable it in the following order.

1. Enable a BFD Port on the VPC

First, enable a BFD Port on the VPC:

```yaml
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 runs in Active-Backup mode.

> **TIP**
>
> The Vpc resource *ovn-cluster* exists by default. You can edit it directly to enable the BFD Port.

After the BFD Port is enabled, a dedicated BFD LRP is automatically created on the 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.

2. Enable BFD on the VPC Egress Gateway

Then enable BFD on the VPC Egress Gateway by setting *.spec.bfd.enabled* to *true*:

```yaml
apiVersion: kubeovn.io/v1
kind: VpcEgressGateway
metadata:
  name: gateway2
  namespace: default
spec:
  vpc: ovn-cluster (1)
  replicas: 2
  internalSubnet: ovn-default (2)
  externalSubnet: underlay-ext (3)
  externalIPs: (4)
    - 172.17.0.11
    - 172.17.0.12
    - 172.17.0.13
  bfd:
    enabled: true (5)
    minRX: 100 (6)
    minTX: 100 (7)
    multiplier: 5 (8)
  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. External IPs assigned to the Egress Gateway instances.
5. Whether to enable BFD for the Egress Gateway.
6. Minimum receive interval for BFD, in milliseconds.
7. Minimum transmit interval for BFD, in milliseconds.

8 Multiplier for BFD, which determines the number of missed packets before declaring a failure.

This example creates a VPC Egress Gateway named *gateway2* with two replicas and BFD enabled. If one instance fails, the BFD session goes down, OVN marks the route as unavailable, and redirects traffic to the healthy instance.

Failover detection time depends on the BFD settings. Use the following formula: *break time = (multiplier + 1) * max(minRX, minTX)*. With this sample configuration, failover detection is approximately 500-600 ms.

In environments with poor network quality or frequent link jitter, a 100 ms BFD interval might be too sensitive. You can increase *.spec.bfd.minRX* and *.spec.bfd.minTX* to 500-1000 ms to reduce false failure detection and frequent failover. Increasing these values also increases the failover detection time, so tune them based on the required balance between stability and failure detection latency.

> **NOTE**
>
> Existing connections may be interrupted during failover and require reconnection. New connections can still be established normally.

3. Verify BFD status

Check the VPC Egress Gateway status:

```
$ kubectl get veg -n default gateway2 -o wide
NAME       VPC           REPLICAS   BFD ENABLED   EXTERNAL SUBNET   PHASE
READY   INTERNAL IPS                EXTERNAL IPS                   WOR
KING NODES       AGE
gateway2   ovn-cluster   2          true          underlay-ext      Compl
eted   true    ["10.16.0.12","10.16.0.13"]   ["172.17.0.11","172.17.0.1
2"]   ["node1","node2"]   58s

$ kubectl get pod -n default -l ovn.kubernetes.io/vpc-egress-gateway=gate
way2 -o wide
NAME                        READY   STATUS    RESTARTS   AGE     IP
NODE    NOMINATED NODE   READINESS GATES
gateway2-fcc6b8b87-8lgvx   1/1     Running   0          2m18s   10.16.0.1
3   node2   <none>           <none>
gateway2-fcc6b8b87-wmww6   1/1     Running   0          2m18s   10.16.0.1
2   node1   <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  1
00.64.0.4
     30000                              ip4.dst == 172.18.0.3  reroute  1
00.64.0.3
     30000                              ip4.dst == 172.18.0.4  reroute  1
00.64.0.2
     29100                    ip4.src == $VEG.8ca38ae7da18.ipv4  reroute  1
0.16.0.12, 10.16.0.13  bfd
     29100                     ip4.src == $VEG.8ca38ae7da18_ip4  reroute  1
0.16.0.12, 10.16.0.13  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  1
00.64.0.3
     29000       ip4.src == $ovn.default.kube.ovn.worker2_ip4  reroute  1
00.64.0.2
     29000        ip4.src == $ovn.default.kube.ovn.worker_ip4  reroute  1
00.64.0.4
     29000     ip4.src == $subnet1.kube.ovn.control.plane_ip4  reroute  1
00.64.0.3
     29000           ip4.src == $subnet1.kube.ovn.worker2_ip4  reroute  1
```

```
00.64.0.2
     29000            ip4.src == $subnet1.kube.ovn.worker_ip4  reroute  1
00.64.0.4

$ kubectl ko nbctl list bfd
_uuid               : 223ede10-9169-4c7d-9524-a546e24bfab5
detect_mult         : 5
dst_ip              : "10.16.0.12"
external_ids        : {af="4", vendor=kube-ovn, vpc-egress-gateway="defau
lt/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              : "10.16.0.13"
external_ids        : {af="4", vendor=kube-ovn, vpc-egress-gateway="defau
lt/gateway2"}
logical_port        : "bfd@ovn-cluster"
min_rx              : 100
min_tx              : 100
options             : {}
status              : up
```

Then check the BFD sessions:

```
$ kubectl exec -n default gateway2-fcc6b8b87-8lgvx -c bfdd -- bfdd-contro
l status
There are 1 sessions:
Session 1
 id=1 local=10.16.0.13 (p) remote=10.255.255.255 state=Up

$ kubectl exec -n default gateway2-fcc6b8b87-wmww6 -c bfdd -- bfdd-contro
l status
There are 1 sessions:
Session 1
 id=1 local=10.16.0.12 (p) remote=10.255.255.255 state=Up
```

> **NOTE**
>
> If all gateway instances are down, egress traffic handled by the VPC Egress Gateway is dropped.

Validated Capacity Guidance

The following results were validated on ACP 4.3.2 with Kube-OVN v1.15.13, two replicas for each VPC Egress Gateway, and bfdd CPU request/limit `50m/200m`. Use them as sizing guidance, not as a hard product limit.

> **NOTE**
>
> The validated capacity is for reference only. Actual capacity can vary with hardware performance and cluster load.

BFD setting	Validated scale	Recommendation
`1000ms / 1000ms x 5`	100 gateways / 200 gateway Pods	Suitable for smaller deployments.
`1500ms / 1500ms x 5`	150 gateways / 300 gateway Pods	Use 1500 ms probing with a detection multiplier of 5.
`3000ms / 3000ms x 5`	300 gateways / 600 gateway Pods	Suitable for medium scale.
`4000ms / 4000ms x 5`	400 gateways / 800 gateway Pods	Keep capacity headroom when using this setting.
`5000ms / 5000ms x 5`	500 gateways / 1000 gateway Pods	Recommended for 500-gateway deployments.

For dedicated Egress Gateway nodes with about 32 vCPUs and 96 GiB memory, the `500ms / 500ms x3` BFD setting and bfdd `50m/200m` CPU request/limit validated up to about 200 gateway Pods per node.

Operations That May Interrupt Traffic

The following operations may briefly interrupt egress traffic because they delete or recreate gateway instances:

1. Changing the number of replicas
2. Changing configuration such as internal or external IPs, node selectors, or BFD settings
3. Upgrading or downgrading Kube-OVN if *.spec.image* is not specified
4. Manually deleting an Egress Gateway Pod

Additional Resources

- RFC 5880 - Bidirectional Forwarding Detection (BFD)↗

Configuring Kube-OVN Network to Support Pod Multi-Network Interfaces

This guide describes how to configure Kube-OVN so that a Pod can use multiple network interfaces.

With Multus CNI and Kube-OVN, you can attach a secondary network to a Pod and manage IP addresses for that network through Kube-OVN Subnet and IPAM.

TIP

- IPv4, IPv6 and dual-stack networks are all supported for both primary and secondary network interfaces.
- Both underlay and overlay networks are supported as secondary networks.

TOC

Before You Begin
Installing Multus CNI
Deploying the Multus CNI Plugin
Create the Secondary Network
Create a Pod with Multiple Network Interfaces
Verify the Network Interfaces
Method 1: Check Pod Annotations

Method 2: Check Network Interfaces from Inside the Pod
Method 3: Check Kube-OVN IP Resources
Optional Configurations
Fixed IP
Additional Routes
Configure Additional Routes for All Pods Referencing the Same NetworkAttachmentDefinition
Configure Additional Routes for a Specific Pod/Workload
Verify Additional Routes in the Pod Route Table
Trademark Notice

Before You Begin

Make sure the following conditions are met:

- The cluster uses Kube-OVN as the network plugin.
- You have permissions to install cluster plugins and create Kubernetes resources.
- `kubectl` is configured and can access the target cluster.

Installing Multus CNI

Deploying the Multus CNI Plugin

1. Navigate to **Administrator** > **Marketplace** > **Cluster Plugins**.
2. Search "multus" and select **"Alauda Container Platform Multi-Network"**.
3. If the plugin is not installed, click the three dots (⋮) on the right and select **Install**.
4. Wait for the plugin state to change to "Installed".

> **NOTE**

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

Create the Secondary Network

To add a secondary network interface to a Pod, first create a NetworkAttachmentDefinition for that network.

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.

The NetworkAttachmentDefinition name MUST NOT contain a dot (`.`). Each NetworkAttachmentDefinition can be used by only one subnet. Do not reuse the same NetworkAttachmentDefinition across multiple subnets.

If you use an underlay secondary network, set `spec.vlan` in the Subnet to the target VLAN CR name.

```
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"
    }
```

Apply the resource:

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

Then create a Kube-OVN subnet for the secondary network interface.

Example: `subnet.yml`

> **NOTE**
>
> `spec.provider` MUST be consistent with the provider in NetworkAttachmentDefinition.

```
apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
  name: subnet1
spec:
  cidrBlock: 172.22.0.0/16
  provider: attachnet.default.ovn
```

Apply the resource:

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

Create a Pod with Multiple Network Interfaces

After the secondary network is ready, create a Pod and reference the NetworkAttachmentDefinition in `metadata.annotations`.

> **NOTE**
>
> - The annotation `k8s.v1.cni.cncf.io/networks` uses the format `<NAMESPACE>/<NAME>`, where `<NAMESPACE>` and `<NAME>` are the namespace and name of the corresponding NetworkAttachmentDefinition.

- To attach multiple secondary networks, specify multiple values separated by commas, for example `default/attachnet,default/attachnet2`.

```yaml
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
```

For workloads that require multiple network interfaces, you can also specify the network attachment annotation in the Pod template of a higher-level controller such as Deployment or StatefulSet. Example:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: web
  template:
    metadata:
      labels:
        app: web
      annotations:
        k8s.v1.cni.cncf.io/networks: default/attachnet
    spec:
      containers:
        - name: web
          image: nginx:latest
          ports:
            - containerPort: 80
```

Verify the Network Interfaces

After the Pod is running, verify the result by using the following methods.

Method 1: Check Pod Annotations

Run the following command:

```bash
kubectl get pod pod1 -o yaml
```

Check the relevant annotations in the output:

```
metadata:
  annotations:
    attachnet.default.ovn.kubernetes.io/allocated: "true"
    attachnet.default.ovn.kubernetes.io/cidr: 172.22.0.0/16
    attachnet.default.ovn.kubernetes.io/gateway: 172.22.0.1
    attachnet.default.ovn.kubernetes.io/ip_address: 172.22.0.23
    attachnet.default.ovn.kubernetes.io/logical_router: ovn-cluster
    attachnet.default.ovn.kubernetes.io/logical_switch: subnet1
    attachnet.default.ovn.kubernetes.io/mac_address: 02:66:77:88:99:aa
    attachnet.default.ovn.kubernetes.io/pod_nic_type: veth-pair
    attachnet.default.ovn.kubernetes.io/routed: "true"
    k8s.v1.cni.cncf.io/network-status: |-
      [{
          "name": "kube-ovn",
          "interface": "eth0",
          "ips": [
              "10.24.0.18/16"
          ],
          "mac": "02:11:22:33:44:55",
          "default": true,
          "dns": {},
          "gateway": [
              "10.24.0.1"
          ]
      },{
          "name": "default/attachnet",
          "interface": "net1",
          "ips": [
              "172.22.0.23"
          ],
          "mac": "02:66:77:88:99:aa",
          "dns": {}
      }]
    k8s.v1.cni.cncf.io/networks: default/attachnet
    ovn.kubernetes.io/allocated: "true"
    ovn.kubernetes.io/cidr: 10.24.0.0/16
    ovn.kubernetes.io/gateway: 10.24.0.1
    ovn.kubernetes.io/ip_address: 10.24.0.18/16
    ovn.kubernetes.io/logical_router: ovn-cluster
    ovn.kubernetes.io/logical_switch: ovn-default
    ovn.kubernetes.io/mac_address: 02:11:22:33:44:55
    ovn.kubernetes.io/pod_nic_type: veth-pair
    ovn.kubernetes.io/routed: "true"
```

Annotation `k8s.v1.cni.cncf.io/network-status` shows the status of all network interfaces of this Pod, including the primary interface and secondary interfaces. In this example, the Pod has two network interfaces: `eth0` is the primary interface, and `net1` is the secondary interface.

Method 2: Check Network Interfaces from Inside the Pod

Run one of the following commands:

```
kubectl exec pod1 -- ifconfig
```

```
kubectl exec pod1 -- ip addr show
```

> **NOTE**
>
> The command availability depends on the container image. If one command is not available in the image, use the other one.

Example output of `ip addr show`:

```
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 02:11:22:33:44:55 brd ff:ff:ff:ff:ff:ff
    inet 10.24.0.18/16 brd 10.24.255.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::11:22ff:fe33:4455/64 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 02:66:77:88:99:aa brd ff:ff:ff:ff:ff:ff
    inet 172.22.0.23/16 brd 172.22.255.255 scope global net1
       valid_lft forever preferred_lft forever
    inet6 fe80::66:77ff:fe88:99aa/64 scope link
       valid_lft forever preferred_lft forever
```

Method 3: Check Kube-OVN IP Resources

Run the following command to verify IP allocation records for both interfaces:

```
kubectl get ip | grep -w pod1.default
```

Example output:

```
NAME                                V4IP         V6IP  MAC
NODE    SUBNET
pod1.default                        10.24.0.18         02:11:22:33:44:55
node-1  ovn-default
pod1.default.attachnet.default.ovn  172.22.0.23        02:66:77:88:99:aa
node-1  subnet1
```

Optional Configurations

Fixed IP

- **Primary network interface**: To assign a fixed IP address to the primary interface, add the annotation `ovn.kubernetes.io/ip_address=<IP>` to the Pod.
- **Secondary network interface**: To assign a fixed IP address to a secondary interface, replace `ovn` in the annotation key with the corresponding NetworkAttachmentDefinition provider. Example: `attachnet.default.ovn.kubernetes.io/ip_address=172.22.0.101`.

> **TIP**
>
> For workloads such as Deployment or StatefulSet, you can specify the fixed IP annotation in the Pod template.

Additional Routes

You can configure additional routes for all Pods that reference the same NetworkAttachmentDefinition or a specific Pod/Workload.

Configure Additional Routes for All Pods Referencing the Same NetworkAttachmentDefinition

To configure additional routes for all secondary network interfaces that reference the same NetworkAttachmentDefinition, add the `routes` field in the NetworkAttachmentDefinition config. 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": "192.168.10.0/24",
        "gw": "172.22.0.101"
      }, {
        "dst": "192.168.20.0/24",
        "gw": "172.22.0.102"
      }]
    }
```

NOTE

Replace the route gateway values with valid addresses that are reachable from the secondary network in your environment.

Configure Additional Routes for a Specific Pod/Workload

To configure additional routes for a specific secondary network interface, add the annotation `attachnet.default.ovn.kubernetes.io/routes` to the Pod. Example:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod1
  annotations:
    k8s.v1.cni.cncf.io/networks: default/attachnet
    attachnet.default.ovn.kubernetes.io/routes: |-
      [{
        "dst": "192.168.10.0/24",
        "gw": "172.22.0.101"
      }]
spec:
  containers:
    - name: web
      image: nginx:latest
      ports:
        - containerPort: 80
```

For workloads such as Deployment or StatefulSet, you can also specify the routes annotation in the Pod template. Example:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: web
  template:
    metadata:
      labels:
        app: web
      annotations:
        k8s.v1.cni.cncf.io/networks: default/attachnet
        attachnet.default.ovn.kubernetes.io/routes: |-
          [{
            "dst": "192.168.10.0/24",
            "gw": "172.22.0.101"
          }]
    spec:
      containers:
        - name: web
          image: nginx:latest
          ports:
            - containerPort: 80
```

Verify Additional Routes in the Pod Route Table

After configuring routes, verify the effective route entries inside the Pod:

```
kubectl exec pod1 -- ip route show
```

Or, if `ip` is not available in the container image, use `route -n`:

```
kubectl exec pod1 -- route -n
```

Example output of `ip route show`:

```
default via 10.24.0.1 dev eth0
10.24.0.0/16 dev eth0 proto kernel scope link src 10.24.0.2
172.22.0.0/16 dev net1 proto kernel scope link src 172.22.0.101
192.168.10.0/24 via 172.22.0.101 dev net1
```

If the corresponding destination and gateway route does not appear in the output, check the `routes` configuration in NetworkAttachmentDefinition or Pod annotations, and then recreate the Pod to re-apply CNI configuration.

Trademark Notice

NGINX® is a registered trademark of F5, Inc. in the U.S. and in other countries.

Alauda is an independent vendor. This product is not affiliated with, endorsed by, or sponsored by F5, Inc. All trademarks are the property of their respective owners and are used here for identification purposes only.

Configure Centralized Gateway

Centralized Gateway allows Pods within a subnet to access the external network using fixed IPs. This is particularly useful for security operations such as network auditing, IP whitelisting, and firewall rule management, where you need to identify and control traffic from specific source IPs. In centralized gateway mode, all outbound traffic from Pods is routed through designated gateway nodes, enabling centralized network policy enforcement and monitoring.

> **NOTE**
>
> Pods under a centralized subnet cannot be accessed through `hostport` or a NodePort type Service with `externalTrafficPolicy: Local`.

If you want traffic within the Subnet to access the external network using a fixed IP for security operations such as auditing and whitelisting, you can set the gateway type in the Subnet to centralized. In centralized gateway mode, packets from Pods accessing the external network are first routed to the `ovn0` NIC of a specific nodes, and then outbound through the host's routing rules. When `natOutgoing` is `true`, the Pod will use the IP of a specific nodes when accessing the external network.

The centralized gateway example is as follows, where the `gatewayType` field is `centralized` and `gatewayNode` is the NodeName of the particular machine in Kubernetes.

```yaml
apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
  name: centralized
spec:
  protocol: IPv4
  cidrBlock: 10.166.0.0/16
  default: false
  excludeIps:
  - 10.166.0.1
  gateway: 10.166.0.1
  gatewayType: centralized
  gatewayNode: "node1,node2"
  natOutgoing: true
```

- If a centralized gateway wants to specify a specific NIC of a machine for outbound networking, `gatewayNode` format can be changed to `kube-ovn-worker:172.18.0.2,kube-ovn-control-plane:172.18.0.3`.
- The spec field `enableEcmp` has been added to the subnet crd definition since Kube-OVN v1.12.0 to migrate the ECMP switch to the subnet level. You can set whether to enable ECMP mode based on different subnets. The `enable-ecmp` parameter in the `kube-ovn-controller` deployment is no longer used. After the previous version is upgraded to v1.12.0, the subnet switch will automatically inherit the value of the original global switch parameter.

> **NOTE**
>
> In centralized gateway ECMP mode, kube-ovn-controller actively probes node status through ping, detecting failures within 5s and completing failover within 5s-10s, during which some traffic may fail.
>
> In centralized gateway primary-backup mode, failover is based on Node Ready status, and it may take several minutes to complete failover in case of power outage.

TOC

Using Label Selectors to Specify Gateway Nodes

In addition to specifying node names directly, you can use `gatewayNodeSelectors` to dynamically select gateway nodes using label selectors. This approach is more flexible, especially useful when node names are not fixed or when you need to select gateways based on labels dynamically.

NOTE

- If `gatewayNode` is not empty, it takes precedence and `gatewayNodeSelectors` is ignored.
- Multiple selectors are evaluated with OR logic - a node matching any selector becomes a gateway node.
- When node labels change, the system automatically updates the gateway node list.

```yaml
apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
  name: centralized-selector
spec:
  protocol: IPv4
  cidrBlock: 10.166.0.0/16
  default: false
  excludeIps:
  - 10.166.0.1
  gateway: 10.166.0.1
  gatewayType: centralized
  gatewayNodeSelectors:
    - matchLabels:
        role: gateway
    - matchExpressions:
        - key: node-type
          operator: In
          values: ["gateway", "egress"]
  natOutgoing: true
```

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
- Trademark Notice

Instructions

Create IPPool

Below is an example:

```yaml
apiVersion: kubeovn.io/v1
kind: IPPool
metadata:
  name: pool-1
spec:
  subnet: ovn-default (1)
  ips: (2)
  - "10.16.0.201"
  - "10.16.0.210/30"
  - "10.16.0.220..10.16.0.230"
  namespaces: (3)
  - ns-1
```

(1) Subnet to which the IP pool belongs.

(2) IP ranges. Supported formats: <*IP*>, <*CIDR*> and <*IP1*>..<*IP2*>. Both IPv4 and IPv6 are supported.

(3) 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:

```yaml
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.:

```yaml
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.

Trademark Notice

NGINX® is a registered trademark of F5, Inc. in the U.S. and in other countries.

Alauda is an independent vendor. This product is not affiliated with, endorsed by, or sponsored by F5, Inc. All trademarks are the property of their respective owners and are used here for identification purposes only.

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 BGP

Kube-OVN can advertise Pod and Subnet routes to external networks using BGP. To use this feature, install `kube-ovn-speaker` on selected (or all) nodes and add the `ovn.kubernetes.io/bgp` annotation to Pods or Subnets that should be reachable from outside the cluster.

`kube-ovn-speaker` uses GoBGP ↗ to publish routes and sets the next hop to itself.

TOC

Installing `kube-ovn-speaker`

Publishing Pod and Subnet routes

Announcement policies

BGP advanced options

Installing `kube-ovn-speaker`

Nodes that run `kube-ovn-speaker` must carry return traffic. Label the nodes where you want to deploy the speaker:

```
kubectl label nodes speaker-node-1 ovn.kubernetes.io/bgp=true
kubectl label nodes speaker-node-2 ovn.kubernetes.io/bgp=true
```

> **NOTE**
>
> When multiple `kube-ovn-speaker` instances are running, each publishes routes upstream. The upstream router must support multi-path ECMP.

Download `speaker.yaml` from a tag or branch that matches your cluster's Kube-OVN version (align with your `kube-ovn` image tag), for example:

```
wget https://raw.githubusercontent.com/kubeovn/kube-ovn/release-1.15/yamls/speaker.yaml
```

Adjust the BGP arguments in the manifest as needed.

If you have one upstream peer:

```
- --neighbor-address=10.32.32.254
- --neighbor-ipv6-address=2409:AB00:AB00:2000::AFB:8AFE
- --neighbor-as=65030
- --cluster-as=65000
```

If you have a pair of switches:

```
- --neighbor-address=10.32.32.252,10.32.32.253
- --neighbor-ipv6-address=2409:AB00:AB00:2000::AFB:8AFC,2409:AB00:AB00:2000::AFB:8AFD
- --neighbor-as=65030
- --cluster-as=65000
```

- `neighbor-address`: BGP peer address, usually the router gateway.
- `neighbor-as`: Autonomous system number of the BGP peer.
- `cluster-as`: Autonomous system number for the container network.

Apply the manifest:

```
kubectl apply -f speaker.yaml
```

Publishing Pod and Subnet routes

For BGP export on a subnet, set `natOutgoing` to `false` on that Subnet so Pod IPs can be used directly on the underlay.

Enable route advertisement with annotations:

```
kubectl annotate pod sample ovn.kubernetes.io/bgp=true
kubectl annotate subnet ovn-default ovn.kubernetes.io/bgp=true
```

Remove the annotations to stop advertising:

```
kubectl annotate pod sample ovn.kubernetes.io/bgp-
kubectl annotate subnet ovn-default ovn.kubernetes.io/bgp-
```

See Announcement policies for behavior when the annotation uses different values.

Announcement policies

`kube-ovn-speaker` supports two policies:

- **Cluster**: Pod IPs and Subnet CIDRs are announced from every speaker, even if no Pod with that IP runs on that node. External traffic may enter any node that hosts a speaker and then be forwarded inside the cluster. Extra hops are possible. This is the default for Pods and Subnets.
- **Local**: Pod IPs are announced only from speakers on nodes that actually host those Pods (or Pods whose Subnet is marked for BGP). External traffic lands on the node where the workload runs, which shortens the path.

> **NOTE**
>
> For the **Local** policy, you typically need `kube-ovn-speaker` on every node. If a Pod lands on a node without a speaker, its IP will not be advertised.

Override the policy per Pod or Subnet with the `ovn.kubernetes.io/bgp` annotation:

- `ovn.kubernetes.io/bgp=cluster`, or the default `ovn.kubernetes.io/bgp=yes`, selects the **Cluster** policy.
- `ovn.kubernetes.io/bgp=local` selects the **Local** policy.

BGP advanced options

`kube-ovn-speaker` supports additional flags for complex environments:

- `auth-password` : Password for the BGP peer.
- `holdtime` : BGP hold time; neighbors with no traffic after this interval are removed (default 90 seconds).
- `graceful-restart` : Enable BGP Graceful Restart.
- `graceful-restart-time` : Graceful Restart time (RFC 4724 section 3).
- `graceful-restart-deferral-time` : Graceful Restart deferral time (RFC 4724 section 4.1).
- `passivemode` : Speaker accepts connections only (passive mode).
- `ebgp-multihop` : TTL for EBGP multi-hop peers (default 1).

Configure IPv4 Multicast for Kube-OVN Underlay Networks

This guide describes how to enable IPv4 multicast communication for workloads in a Kube-OVN Underlay subnet. After multicast snooping is enabled for a subnet, Kube-OVN configures multicast snooping and a multicast querier on the corresponding OVN logical switch.

This configuration applies to each subnet independently. It does not configure multicast routing between different Layer 3 networks or configure vendor-specific physical switches.

TOC

Before You Begin
Enable Multicast Snooping
Using the Web Console
View the Configuration Result
Disable Multicast Snooping

Before You Begin

Make sure the following conditions are met:

- **ACP version**: 4.2

- The target subnet is a Kube-OVN Underlay subnet with `spec.protocol` set to `IPv4`. IPv6 and dual-stack subnets are outside the scope of this guide.
- The physical network has been prepared according to Preparing Kube-OVN Underlay Physical Network.
- The subnet has at least one available IP address. When multicast snooping is enabled, Kube-OVN creates a multicast querier for the subnet. The querier uses one available IP address from the subnet address pool and an automatically generated MAC address.
- The physical network permits the multicast group and IGMP traffic used by the application.

If IGMP snooping is enabled on the physical switches, work with the network administrator to verify that multicast membership can be learned on the VLAN ports connected to the cluster nodes and external receivers. The querier and IGMP snooping behavior of physical switches varies by vendor.

Enable Multicast Snooping

Using the Web Console

1. Go to **Administrator**.
2. From the **Cluster** drop-down list at the top, select the target cluster.
3. In the left navigation bar, expand **Clusters**, and then click **Resources**.
4. In the resource search box, enter **Subnet**. Expand **Correlated with Cluster**, and then select **Subnet** (`kubeovn.io`).
5. Click the action menu (⋮) next to the target Underlay subnet, and then select **Update**.
6. Add `enableMulticastSnoop: true` to `spec`:

```
spec:
  enableMulticastSnoop: true
```

7. Click **Update**.

Repeat these steps for every Underlay subnet that needs to carry multicast traffic.

View the Configuration Result

After the update is complete, view the YAML of the target subnet in the web console and verify the following fields:

- `spec.enableMulticastSnoop` is `true`, indicating that multicast snooping is enabled.
- `status.mcastQuerierIP` and `status.mcastQuerierMAC` have values, indicating that the multicast querier has been created.

If the querier IP or MAC address is empty, make sure the subnet has an available IP address.

Disable Multicast Snooping

In the web console, open the YAML of the target subnet, set `spec.enableMulticastSnoop` to `false`, and then click **Update**.

After the update is complete, Kube-OVN releases the querier IP and MAC address, and `status.mcastQuerierIP` and `status.mcastQuerierMAC` become empty.

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
- Trademark Notice

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
Node Selectors	Configure label selectors to specify which nodes the Endpoint Health Checker components should run on. Click **Add** to add multiple label key-value pairs.

Parameter	Description
Node Tolerations	Configure tolerations to allow Endpoint Health Checker components to be scheduled on nodes with specific taints. Click **Add** 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`

```yaml
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`.

```yaml
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

Follow the instructions in How To Add Pod Annotations In Envoy Gateway, setting the annotation to `endpoint-health-checker.io/enabled: 'true'`.

For Custom Deployment

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

```yaml
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:

```yaml
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.

Trademark Notice

Envoy® is a registered trademark of The Linux Foundation.

NGINX® is a registered trademark of F5, Inc. in the U.S. and in other countries.

MetalLB™ is a trademark of The Linux Foundation.

Alauda is an independent vendor. This product is not affiliated with, endorsed by, or sponsored by The Linux Foundation or F5, Inc. All trademarks are the property of their respective owners and are used here for identification purposes only.

Tasks for ALB

TOC

How To Set NodeSelector And Tolerations For alb-operator

How To Set NodeSelector And Tolerations For alb

Trademark Notice

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-operat
or    --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": ""
      }
    }
  }
}'
```

Trademark Notice

NGINX® is a registered trademark of F5, Inc. in the U.S. and in other countries.

Alauda is an independent vendor. This product is not affiliated with, endorsed by, or sponsored by F5, Inc. All trademarks are the property of their respective owners and are used here for identification purposes only.

Task: Migrate from OCP Route to GatewayAPI Route

TOC

- Introduction
- Prerequisites
- Basic HTTP Route
  - OCP Route Configuration
  - Gateway API Configuration
- Route Timeouts
  - OCP Route Configuration
  - Gateway API Configuration
- HTTP Strict Transport Security (HSTS)
  - OCP Route Configuration
  - Gateway API Configuration
- Cookie-Based Session Affinity
  - OCP Route Configuration
  - Gateway API Configuration
- Path-Based Routing
  - OCP Route Configuration
  - Gateway API Configuration
- Header Modification
  - OCP Route Configuration

Gateway API Configuration

Connection Limits

OCP Route Configuration

Gateway API Configuration

Rate Limiting

OCP Route Configuration

Gateway API Configuration

IP Allowlist/Blocklist

OCP Route Configuration

Gateway API Configuration

URL Rewrite

OCP Route Configuration

Gateway API Configuration

Cross-Namespace Route Admission

OCP Route Configuration

Gateway API Configuration

Default TLS Certificate for Ingress

OCP Route Configuration

Gateway API Configuration

TLS Re-encrypt with Custom CA

OCP Route Configuration

Gateway API Configuration

Edge Termination with Custom Certificate

OCP Route Configuration

Gateway API Configuration

TLS Passthrough

OCP Route Configuration

Gateway API Configuration

Feature Comparison Summary

Migration Strategy

Related Documentation

Trademark Notice

Introduction

Migrate from OpenShift Container Platform (OCP) Routes to Kubernetes Gateway API HTTPRoutes with Envoy Gateway. Use the feature-specific mappings below to translate each OCP Route feature into the equivalent Gateway API configuration.

Prerequisites

1. Configure EnvoyGatewayCtl
2. Configure Gateway
3. Basic understanding of Gateway API Routes

Basic HTTP Route

OCP Route Configuration

In OCP, a basic HTTP route is created using the Route resource:

```yaml
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: example-route
  namespace: demo
spec:
  host: example.com 1
  to: 2
    kind: Service
    name: example-service
  port: 3
    targetPort: 8080
```

1 The hostname for the route

2 Backend service reference

3 Target port on the backend service

Gateway API Configuration

In Gateway API, the equivalent configuration uses HTTPRoute:

```yaml
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: example-route
  namespace: demo
spec:
  hostnames: 1
    - example.com
  parentRefs: 2
    - name: demo-gateway
      namespace: demo
  rules:
    - backendRefs: 3
        - name: example-service
          port: 8080
```

1 Hostnames that this route accepts (equivalent to OCP Route's `host`)

2 Reference to the Gateway listener

3 Backend service and port

Route Timeouts

OCP Route Configuration

In OCP, timeouts are configured using annotations:

```yaml
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: example-route
  annotations:
    haproxy.router.openshift.io/timeout: 30s (1)
    haproxy.router.openshift.io/timeout-tunnel: 1h (2)
spec:
  host: example.com
  to:
    kind: Service
    name: example-service
```

(1) General timeout for HTTP requests

(2) Timeout for tunnel connections (WebSocket, HTTP/2, etc.)

Gateway API Configuration

In Gateway API, timeouts are configured in the HTTPRoute rules:

```yaml
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: example-route
  namespace: demo
spec:
  hostnames:
    - example.com
  parentRefs:
    - name: demo-gateway
  rules:
    - backendRefs:
        - name: example-service
          port: 8080
      timeouts: (1)
        request: 30s
        backendRequest: 25s
```

(1) Request timeout configuration

For more details, see Request Timeouts↗.

> **NOTE**
>
> Gateway API does not have a separate timeout specifically for tunnel connections. The `request` timeout applies to all connection types.

HTTP Strict Transport Security (HSTS)

OCP Route Configuration

In OCP, HSTS is configured using the annotation on edge-terminated or re-encrypt routes:

```yaml
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: example-route
  annotations:
    haproxy.router.openshift.io/hsts_header: max-age=31536000;includeSubD
omains;preload (1)
spec:
  host: example.com
  to:
    kind: Service
    name: example-service
  tls:
    termination: edge
```

(1) HSTS header configuration

Gateway API Configuration

In Gateway API, HSTS is configured using response header modification:

```yaml
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: example-route
  namespace: demo
spec:
  hostnames:
    - example.com
  parentRefs:
    - name: demo-gateway
  rules:
    - filters: (1)
        - type: ResponseHeaderModifier
          responseHeaderModifier:
            add:
              - name: Strict-Transport-Security
                value: max-age=31536000;includeSubDomains;preload
      backendRefs:
        - name: example-service
          port: 8080
```

(1) Add HSTS header to response

For more details, see HTTP Header Modification ↗.

> **NOTE**
>
> Unlike OCP's `requiredHSTSPolicies` cluster-wide enforcement, Gateway API requires explicit configuration per route. There is no global HSTS policy enforcement mechanism in Gateway API. ACP provides Gateway-wide HSTS through the Envoy Gateway `ClientTrafficPolicy` extension; for more information, see How To Configure HSTS.

Cookie-Based Session Affinity

OCP Route Configuration

In OCP, session affinity is configured using annotations:

```yaml
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: example-route
  annotations:
    haproxy.router.openshift.io/balance: source (1)
    haproxy.router.openshift.io/disable_cookies: "false" (2)
    router.openshift.io/cookie_name: my-cookie (3)
spec:
  host: example.com
  to:
    kind: Service
    name: example-service
```

1. Load balancing algorithm
2. Enable cookie-based session affinity
3. Cookie name for session persistence

Gateway API Configuration

In Gateway API, session persistence is configured in the HTTPRoute rules:

```yaml
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: example-route
  namespace: demo
spec:
  hostnames:
    - example.com
  parentRefs:
    - name: demo-gateway
  rules:
    - backendRefs:
        - name: example-service
          port: 8080
      sessionPersistence: 1
        type: Cookie
        sessionName: my-cookie
        cookieConfig:
          lifetimeType: Permanent 2
```

1 Session persistence configuration

2 Cookie lifetime type: `Permanent` (persists across browser sessions) or `Session` (expires when browser closes)

For more details, see Session Affinity/Sticky Sessions↗.

Path-Based Routing

OCP Route Configuration

In OCP, path-based routing uses the `path` field:

```yaml
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: example-route
spec:
  host: example.com
  path: /api (1)
  to:
    kind: Service
    name: example-service
```

(1) Path prefix for matching requests

Gateway API Configuration

In Gateway API, path matching is configured in the HTTPRoute rules:

```yaml
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: example-route
  namespace: demo
spec:
  hostnames:
    - example.com
  parentRefs:
    - name: demo-gateway
  rules:
    - matches: (1)
        - path:
            type: PathPrefix
            value: /api
      backendRefs:
        - name: example-service
          port: 8080
```

(1) Path matching configuration

Gateway API supports multiple match types:

- `PathPrefix`: Matches the path prefix (equivalent to OCP's default behavior)
- `Exact`: Matches the exact path
- `RegularExpression`: Matches using regular expressions

For more details, see HTTPRoute Matches.

Header Modification

OCP Route Configuration

In OCP, header modification uses annotations:

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: example-route
  annotations:
    haproxy.router.openshift.io/response-set-header: X-Custom-Header:valu
e (1)
    haproxy.router.openshift.io/request-set-header: X-Request-Header:valu
e (2)
spec:
  host: example.com
  to:
    kind: Service
    name: example-service
```

(1) Set response headers

(2) Set request headers

Gateway API Configuration

In Gateway API, header modification is configured using filters:

```yaml
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: example-route
  namespace: demo
spec:
  hostnames:
    - example.com
  parentRefs:
    - name: demo-gateway
  rules:
    - filters:
        - type: RequestHeaderModifier (1)
          requestHeaderModifier:
            add:
              - name: X-Request-Header
                value: value
        - type: ResponseHeaderModifier (2)
          responseHeaderModifier:
            add:
              - name: X-Custom-Header
                value: value
      backendRefs:
        - name: example-service
          port: 8080
```

(1) Request header modification

(2) Response header modification

For more details, see HTTP Header Modification ↗.

Connection Limits

OCP Route Configuration

In OCP, connection limits are configured using annotations:

```yaml
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: example-route
  annotations:
    haproxy.router.openshift.io/pod-concurrent-connections: "100" (1)
spec:
  host: example.com
  to:
    kind: Service
    name: example-service
```

(1) Maximum concurrent connections per backend pod

Gateway API Configuration

In Gateway API, connection limits are configured using ClientTrafficPolicy attached to the Gateway:

```yaml
apiVersion: gateway.envoyproxy.io/v1alpha1
kind: ClientTrafficPolicy
metadata:
  name: connection-limit-policy
  namespace: demo
spec:
  targetRefs: (1)
    - group: gateway.networking.k8s.io
      kind: Gateway
      name: demo-gateway
  connection: (2)
    connectionLimit:
      value: 100
```

(1) Attach policy to Gateway

(2) Connection limit configuration

For more details, see Connection Limit↗.

NOTE

Unlike OCP's per-backend pod limit, Gateway API's connection limit is applied at the Gateway level and is distributed across Envoy proxy instances.

Rate Limiting

OCP Route Configuration

In OCP, rate limiting uses annotations:

```yaml
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: example-route
  annotations:
    haproxy.router.openshift.io/rate-limit-connections: "true" 1
    haproxy.router.openshift.io/rate-limit-connections.concurrent-tcp: "1
0"
    haproxy.router.openshift.io/rate-limit-connections.rate-http: "100"
spec:
  host: example.com
  to:
    kind: Service
    name: example-service
```

1 Rate limiting configuration

Gateway API Configuration

In Gateway API, rate limiting is configured using BackendTrafficPolicy:

```yaml
apiVersion: gateway.envoyproxy.io/v1alpha1
kind: BackendTrafficPolicy
metadata:
  name: rate-limit-policy
  namespace: demo
spec:
  targetRefs:
    - group: gateway.networking.k8s.io
      kind: HTTPRoute
      name: example-route
  rateLimit: 1
    type: Local
    local:
      rules:
        - limit:
            requests: 100
            unit: Second
```

1 Rate limiting configuration

For more details, see Rate Limiting ↗.

> **NOTE**
>
> Gateway API rate limiting is more flexible than OCP Route annotations and supports both local and global rate limiting mechanisms.

IP Allowlist/Blocklist

OCP Route Configuration

In OCP, IP allowlisting uses annotations:

```yaml
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: example-route
  annotations:
    haproxy.router.openshift.io/ip_allowlist: 192.168.1.0/24 10.0.0.1 1
spec:
  host: example.com
  to:
    kind: Service
    name: example-service
```

1 IP allowlist configuration

Gateway API Configuration

In Gateway API, IP filtering is implemented using SecurityPolicy with authorization rules:

```yaml
apiVersion: gateway.envoyproxy.io/v1alpha1
kind: SecurityPolicy
metadata:
  name: ip-filter-policy
  namespace: demo
spec:
  targetRefs: 1
    - group: gateway.networking.k8s.io
      kind: HTTPRoute
      name: example-route
  authorization: 2
    defaultAction: Deny
    rules:
      - action: Allow
        principal:
          clientCIDRs: 3
            - 192.168.1.0/24
            - 10.0.0.1/32
```

1 Attach policy to HTTPRoute

2 Authorization configuration with default deny

3 IP allowlist using CIDR notation

For IP blocklist (denylist), set `defaultAction: Allow` and use `action: Deny`:

```yaml
apiVersion: gateway.envoyproxy.io/v1alpha1
kind: SecurityPolicy
metadata:
  name: ip-blocklist-policy
  namespace: demo
spec:
  targetRefs:
    - group: gateway.networking.k8s.io
      kind: HTTPRoute
      name: example-route
  authorization:
    defaultAction: Allow (1)
    rules:
      - action: Deny (2)
        principal:
          clientCIDRs:
            - 192.168.100.0/24
```

1 Default action allows all traffic

2 Deny traffic from specific IPs

For more details, see IP Allowlist/Denylist↗.

> **NOTE**
>
> Ensure you configure the client IP detection correctly using ClientTrafficPolicy if your Gateway is behind a load balancer or proxy.

URL Rewrite

OCP Route Configuration

In OCP, URL rewriting uses the annotation:

```yaml
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: example-route
  annotations:
    haproxy.router.openshift.io/rewrite-target: /new-path (1)
spec:
  host: example.com
  path: /old-path
  to:
    kind: Service
    name: example-service
```

(1) Rewrite target path

Gateway API Configuration

In Gateway API, URL rewriting is configured using filters:

```yaml
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: example-route
  namespace: demo
spec:
  hostnames:
    - example.com
  parentRefs:
    - name: demo-gateway
  rules:
    - matches:
        - path:
            type: PathPrefix
            value: /old-path
      filters:
        - type: URLRewrite (1)
          urlRewrite:
            path:
              type: ReplacePrefixMatch
              replacePrefixMatch: /new-path
      backendRefs:
        - name: example-service
          port: 8080
```

(1) URL rewrite filter

For more details, see URL Rewrite ↗.

Cross-Namespace Route Admission

OCP Route Configuration

In OCP, the route admission policy controls whether routes in different namespaces can claim the same hostname. This is configured at the cluster level through the Ingress Operator.

Gateway API Configuration

In Gateway API, cross-namespace access is controlled at the Gateway listener level:

```yaml
apiVersion: gateway.networking.k8s.io/v1
kind: Gateway
metadata:
  name: demo-gateway
  namespace: gateway-ns
spec:
  gatewayClassName: envoy-gateway-operator-cpaas-default
  listeners:
    - name: http
      protocol: HTTP
      port: 80
      allowedRoutes: 1
        namespaces:
          from: All  # Allow routes from all namespaces
```

1 Configure which namespaces can attach routes

Options for `allowedRoutes.namespaces.from`:

- `Same`: Only routes in the same namespace as the Gateway
- `All`: Routes from any namespace
- `Selector`: Routes from namespaces matching a label selector

For more details, see Cross-Namespace Routing.

> **NOTE**
>
> Unlike OCP's cluster-wide admission policy, Gateway API controls this at the individual Gateway listener level, providing more granular control.

Default TLS Certificate for Ingress

OCP Route Configuration

In OCP, routes without TLS configuration can use a default certificate configured at the Ingress Controller level.

Gateway API Configuration

In Gateway API, configure a default TLS certificate on the Gateway listener:

```yaml
apiVersion: gateway.networking.k8s.io/v1
kind: Gateway
metadata:
  name: demo-gateway
  namespace: demo
spec:
  gatewayClassName: envoy-gateway-operator-cpaas-default
  listeners:
    - name: https
      protocol: HTTPS
      port: 443
      tls: (1)
        mode: Terminate
        certificateRefs:
          - name: default-tls-cert
```

(1) Default TLS certificate for the listener

Any HTTPRoute attached to this listener without specific TLS configuration will use this default certificate.

TLS Re-encrypt with Custom CA

OCP Route Configuration

In OCP, re-encrypt routes terminate TLS at the router and re-encrypt to the backend with validation:

```yaml
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: example-route
spec:
  host: example.com
  to:
    kind: Service
    name: example-service
  tls:
    termination: reencrypt (1)
    destinationCACertificate: | (2)
      -----BEGIN CERTIFICATE-----
      ...
      -----END CERTIFICATE-----
```

(1) Re-encrypt termination mode

(2) CA certificate for validating backend

Gateway API Configuration

In Gateway API, backend TLS validation is configured using BackendTLSPolicy:

```yaml
apiVersion: gateway.networking.k8s.io/v1
kind: Gateway
metadata:
  name: demo-gateway
  namespace: demo
spec:
  gatewayClassName: envoy-gateway-operator-cpaas-default
  listeners:
    - name: https
      protocol: HTTPS
      port: 443
      tls:
        mode: Terminate (1)
        certificateRefs:
          - name: frontend-tls
---
apiVersion: gateway.networking.k8s.io/v1
kind: BackendTLSPolicy
metadata:
  name: backend-tls-policy
  namespace: demo
spec:
  targetRefs: (2)
    - group: ""
      kind: Service
      name: example-service
  validation: (3)
    caCertificateRefs:
      - name: backend-ca-cert
        kind: ConfigMap
    hostname: backend.example.com
---
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: example-route
  namespace: demo
spec:
  parentRefs:
    - name: demo-gateway
  hostnames:
    - example.com
  rules:
```

```
    - backendRefs:
        - name: example-service
          port: 8443
```

1. Terminate TLS at the Gateway
2. Apply policy to the backend Service
3. Backend TLS validation configuration

For more details, see Backend TLS↗.

> **NOTE**
>
> The CA certificate must be stored in a ConfigMap, not a Secret.

Edge Termination with Custom Certificate

OCP Route Configuration

In OCP, edge termination uses the route's TLS configuration:

```yaml
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: example-route
spec:
  host: example.com
  to:
    kind: Service
    name: example-service
  tls:
    termination: edge (1)
    certificate: | (2)
      -----BEGIN CERTIFICATE-----
      ...
      -----END CERTIFICATE-----
    key: | (3)
      -----BEGIN PRIVATE KEY-----
      ...
      -----END PRIVATE KEY-----
```

(1) Edge termination mode

(2) TLS certificate

(3) TLS private key

Gateway API Configuration

In Gateway API, TLS termination is configured on the Gateway listener:

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: example-tls
  namespace: demo
type: kubernetes.io/tls (1)
data:
  tls.crt: <base64-encoded-cert>
  tls.key: <base64-encoded-key>
---
apiVersion: gateway.networking.k8s.io/v1
kind: Gateway
metadata:
  name: demo-gateway
  namespace: demo
spec:
  gatewayClassName: envoy-gateway-operator-cpaas-default
  listeners:
    - name: https
      protocol: HTTPS (2)
      port: 443
      tls:
        mode: Terminate (3)
        certificateRefs: (4)
          - name: example-tls
---
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: example-route
  namespace: demo
spec:
  parentRefs:
    - name: demo-gateway
      sectionName: https
  hostnames:
    - example.com
  rules:
    - backendRefs:
        - name: example-service
          port: 8080
```

1. TLS secret type
2. HTTPS protocol
3. Terminate TLS mode
4. Reference to TLS secret

TLS Passthrough

OCP Route Configuration

In OCP, passthrough routes do not terminate TLS:

```yaml
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: example-route
spec:
  host: example.com
  to:
    kind: Service
    name: example-service
  tls:
    termination: passthrough (1)
```

1. Passthrough termination mode

Gateway API Configuration

In Gateway API, TLS passthrough uses TLSRoute:

```yaml
apiVersion: gateway.networking.k8s.io/v1
kind: Gateway
metadata:
  name: demo-gateway
  namespace: demo
spec:
  gatewayClassName: envoy-gateway-operator-cpaas-default
  listeners:
    - name: tls-passthrough
      protocol: TLS (1)
      port: 443
      tls:
        mode: Passthrough (2)
---
apiVersion: gateway.networking.k8s.io/v1alpha2
kind: TLSRoute
metadata:
  name: example-route
  namespace: demo
spec:
  parentRefs:
    - name: demo-gateway
      sectionName: tls-passthrough
  hostnames: (3)
    - example.com
  rules:
    - backendRefs:
        - name: example-service
          port: 8443
```

(1) TLS protocol

(2) Passthrough mode

(3) SNI hostname for routing

For more details, see TLS Passthrough and TLSRoute specification ↗.

NOTE

TLSRoute uses SNI (Server Name Indication) for routing decisions without terminating TLS. This means:

- No access to HTTP headers or request content for routing decisions
- No L7 filters (header modification, URL rewrite, etc.) can be applied
- Routing is purely based on the SNI hostname

Feature Comparison Summary

Feature	OCP Route	Gateway API
Basic HTTP Routing	Route spec	HTTPRoute
Path-based Routing	`.spec.path`	`.spec.rules[].matches[].path`
Timeouts	Annotations	HTTPRoute `.spec.rules[].timeouts`
HSTS	Annotation	ResponseHeaderModifier filter
Session Affinity	Annotations	HTTPRoute `.spec.rules[].sessionPersistence`
Header Modification	Annotations	RequestHeaderModifier/ResponseHeaderModifier filters
Connection Limits	Annotation	ClientTrafficPolicy

Feature	OCP Route	Gateway API
Rate Limiting	Annotations	BackendTrafficPolicy
IP Allowlist/Blocklist	Annotation	SecurityPolicy with authorization
URL Rewrite	Annotation	URLRewrite filter
Cross-Namespace	Cluster-level policy	Gateway `.spec.listeners[].allowedRoutes`
Default TLS Cert	Controller-level	Gateway listener TLS
TLS Re-encrypt	`.spec.tls.termination: reencrypt`	BackendTLSPolicy
TLS Edge	`.spec.tls.termination: edge`	Gateway listener TLS with HTTPS
TLS Passthrough	`.spec.tls.termination: passthrough`	TLSRoute with Passthrough mode

Migration Strategy

When migrating from OCP Routes to Gateway API:

1. **Start with a Gateway**: Create a Gateway resource with appropriate listeners for your use case
    - Deploy the Gateway in the same namespace or a dedicated gateway namespace

- Configure listeners for all protocols you need (HTTP, HTTPS, TLS, TCP, UDP)
- Ensure the Gateway service is created and has an accessible endpoint

2. **Convert Routes to HTTPRoutes**: Migrate each OCP Route to an HTTPRoute, starting with simple routes
   - Begin with non-production or low-traffic routes to minimize risk
   - Verify hostname and path matching configurations
   - Test basic connectivity before adding advanced features
3. **Apply Policies**: For features that require policies (BackendTrafficPolicy, SecurityPolicy, ClientTrafficPolicy), create and attach them after the basic route is working
   - Add one policy at a time and validate functionality
   - Monitor for any unexpected behavior or performance impact
4. **Test Incrementally**: Validate each migration step before proceeding to the next route
   - **Monitoring**: Check Gateway and Route status conditions for any errors
   - **Functional Testing**: Verify all routes work as expected with curl or automated tests
   - **Performance Testing**: Compare response times and throughput with OCP Routes
   - **Validation Checks**:
     - Verify TLS certificates are correctly applied
     - Test session affinity and load balancing behavior
     - Validate rate limiting and security policies
     - Check header modification and URL rewrites
5. **Update DNS**: Once validated, update DNS entries to point to the new Gateway service
   - **Dual-Running Period** (recommended): Point a subset of traffic to the new Gateway while keeping OCP Routes active for rollback capability
   - Gradually shift traffic using weighted DNS or canary deployments
   - Monitor error rates, latency, and application metrics during cutover
6. **Rollback Procedures**: If issues are discovered after migration
   - Revert DNS to point back to OCP Routes

- Keep OCP Routes active for at least 24-48 hours after DNS cutover
- Document any configuration differences that caused issues
- Have a communication plan for stakeholders if rollback is needed

Related Documentation

- Configure GatewayAPI Gateway
- Configure GatewayAPI Route
- Configure GatewayAPI Policy
- Tasks for Envoy Gateway

Trademark Notice

Envoy® is a registered trademark of The Linux Foundation.

Alauda is an independent vendor. This product is not affiliated with, endorsed by, or sponsored by The Linux Foundation. All trademarks are the property of their respective owners and are used here for identification purposes only.

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.
```