Menu

Networking

Introduction

Architecture

Concepts

Guides

How To

Trouble Shooting

Introduction

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 for managing HTTPS ingress traffic.
- MetalLB for handling LoadBalancer type Services.
- Additionally, it provides robust network security and encryption features to ensure secure communication.

TOC

Usage Limitations

While the container network provides extensive functionalities, the following limitations should be noted:

- **Underlay Network Requirement**

  Some underlay network capabilities, such as Kube-OVN Underlay Subnet, Egress IP, and MetalLB, require underlying L2 network support. These features cannot be used in public cloud providers and certain virtualized environments like AWS and GCP.

With its versatile design and comprehensive feature set, the container network empowers organizations to build, scale, and manage secure, reliable, and high-performance containerized applications.

Menu

Architecture

Understanding Kube-OVN

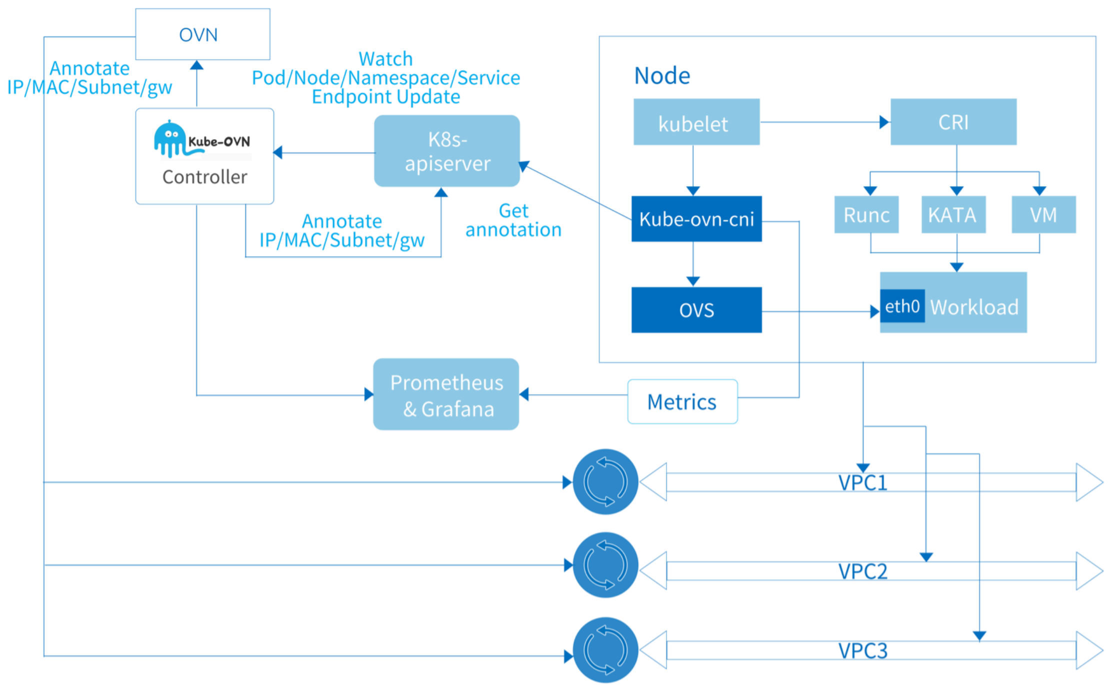
This document describes the general architecture of Kube-OVN, the functionality of each component and how they interact with each other.

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

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

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

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



TOC

Upstream OVN/OVS Components

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

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

ovn-central

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

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

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

ovs-ovn

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

Core Controller and Agent

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

kube-ovn-controller

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

Pod, Service, Endpoint, Node, NetworkPolicy, VPC, Subnet, Vlan, ProviderNetwork.

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

kube-ovn-cni

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

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

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

1. Config `ovn-controller` and `vswitchd`.
2. Handle CNI Add/Del requests:

   2.1. Create or delete veth pair and bind or unbind to OVS ports.

   2.2. Configure OVS ports

   2.3. Update host iptables/ipset/route rules.

3. Dynamically update the network QoS.
4. Create and configure the `ovn0` NIC to connect the container network and the host network.
5. Configure the host NIC to implement Vlan/Underlay/EIP.
6. Dynamically config inter-cluster gateways.

Monitoring, Operation and Maintenance Tools and Extension Components

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

kube-ovn-speaker

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

kube-ovn-pinger

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

kube-ovn-monitor

This component collects OVN status information and the monitoring metrics.

kubectl-ko

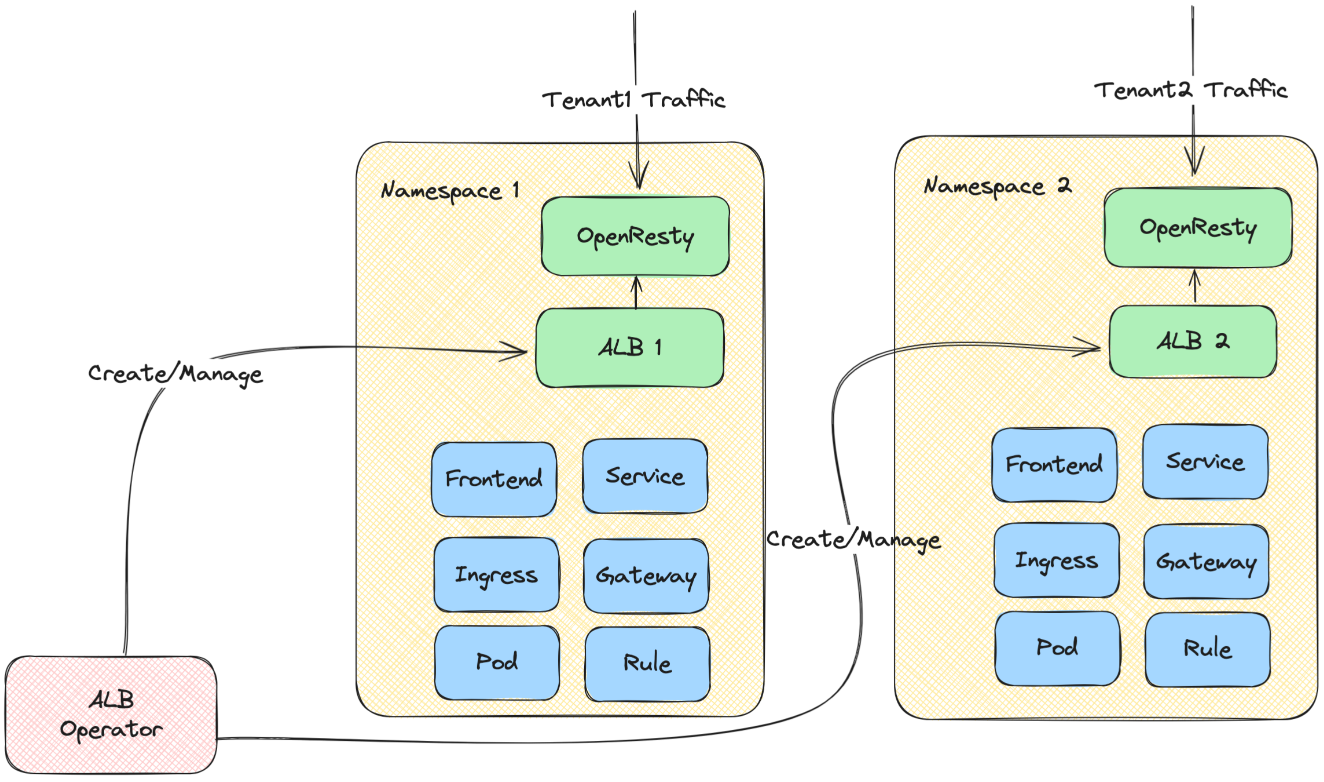
This component is a kubectl plugin, which can quickly run common operations.

Understanding ALB

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

TOC

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: docker.io/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

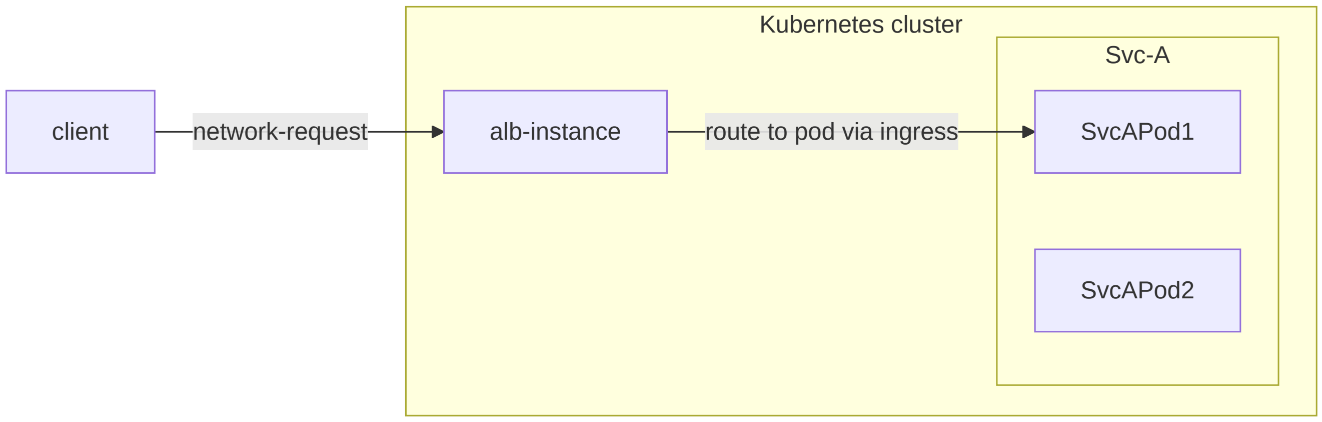
LoadBalancer is a key component in modern cloud-native architectures, serving as an intelligent traffic router and load balancer.

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

Understanding MetalLB

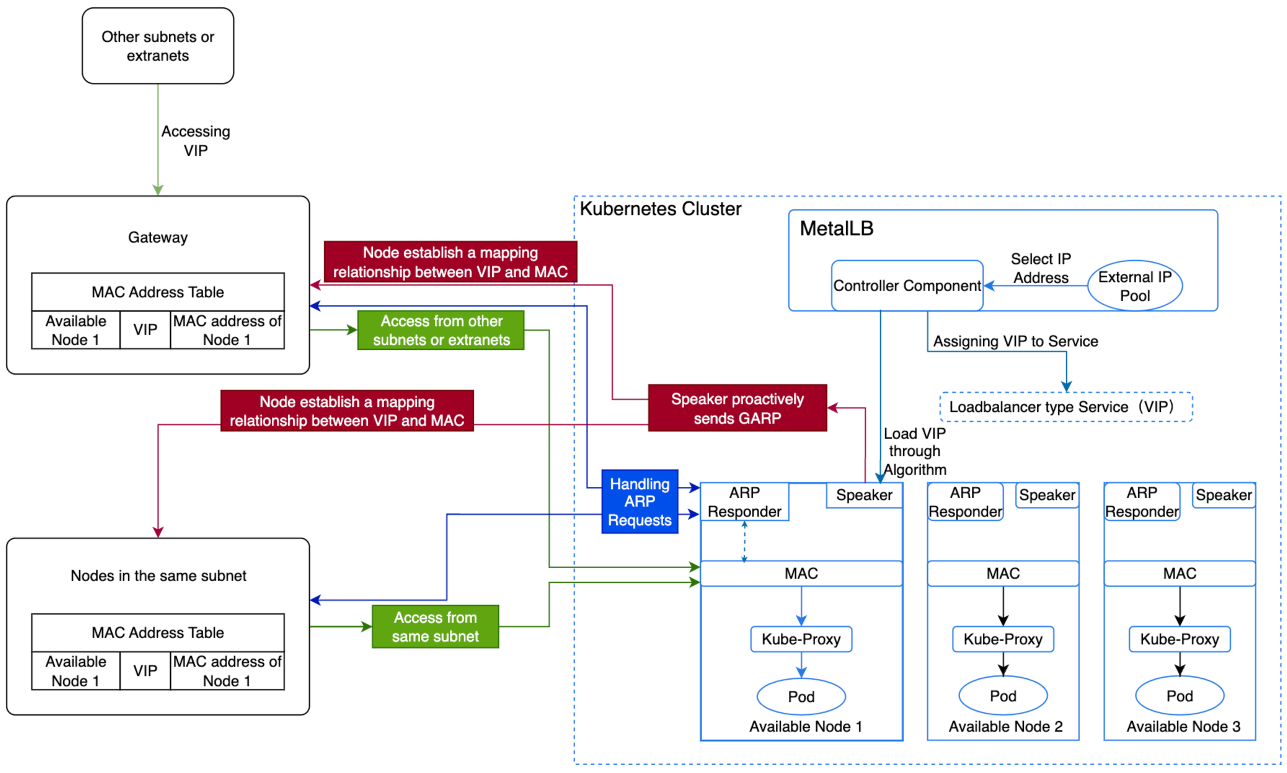
TOC

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

Term	Description
	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 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.

Additional resources

- Creating External IP Address Pool
- Creating BGP Peers

Menu

Concepts

ALB with Ingress-NGINX Annotation Compatibility

TOC

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
nginx.ingress.kubernetes.io/affinity	cookie	o ingress does not support. alb rule can

Name	type	Support (v supports x does not support o partially supports or can be achieved by configuration)
		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
nginx.ingress.kubernetes.io/auth-tls-error-page	string	x
nginx.ingress.kubernetes.io/auth-tls-pass-certificate-to-upstream	"true" or "false"	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-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
nginx.ingress.kubernetes.io/backend-protocol	string	v
nginx.ingress.kubernetes.io/canary	"true" or "false"	x

Name	type	Support (v supports x does not support o partially supports or can be achieved by configuration)
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
nginx.ingress.kubernetes.io/enable-cors	"true" or "false"	v
nginx.ingress.kubernetes.io/cors-allow-origin	string	v

Name	type	Support (v supports x does not support o partially supports or can be achieved by configuration)
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
nginx.ingress.kubernetes.io/global-rate-limit-window	duration	x
nginx.ingress.kubernetes.io/global-rate-limit-key	string	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-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
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

Name	type	Support (v supports x does not support o partially supports or can be achieved by configuration)
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
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

Name	type	Support (v supports x does not support o partially supports or can be achieved by configuration)
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
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

Name	type	Support (v supports x does not support o partially supports or can be achieved by configuration)
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
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

Name	type	Support (v supports x does not support o partially supports or can be achieved by configuration)
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
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

Comparison Among Service, Ingress, Gateway API, and ALB Rule

The Alauda Container Platform supports multiple ingress traffic specifications in Kubernetes ecosystem. This document compares them (Service, Ingress, Gateway API, and ALB Rule) to help users make the right choice.

TOC

For L4(TCP/UDP) Traffic

Services of type LoadBalancer, Gateway API, and ALB Rules can all expose L4 traffic externally. Here we recommend using the LoadBalancer type Service approach. Both Gateway API and ALB Rules are implemented by ALB, which is a userspace proxy, and their performance degrades significantly when handling L4 traffic compared to LoadBalancer type Services.

For L7(HTTP/HTTPS) Traffic

While Ingress, GatewayAPI, and ALB Rules can all expose L7 traffic externally, they differ in their capabilities and isolation models.

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

GatewayAPI provides more flexible isolation mode, however they are not as mature as Ingress. By using GatewayAPI developer can create their own isolated ALB instances to handle GatewayAPI rules. Therefore, if you need to delegate the creation and management of ALB instances to developers, you need to choose to use GatewayAPI.

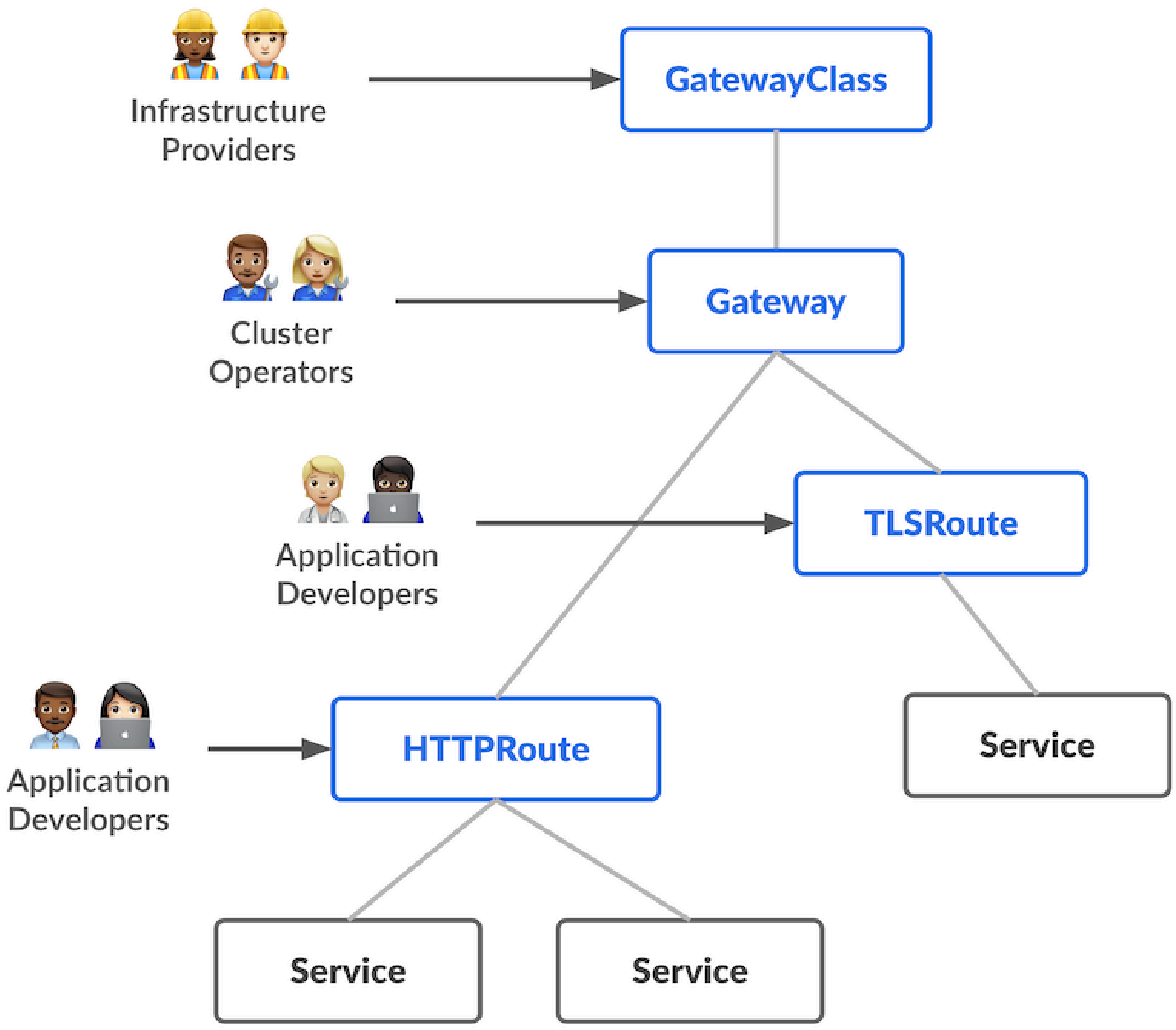
ALB Rule

ALB Rule(Load Balancer in the UI) provides the most flexible traffic match rules and the most capabilities. In fact, both Ingress and GatewayAPI are implemented by translating them to ALB Rules. However, the ALB Rule is more complex than Ingress and GatewayAPI and is not a community-standard API. Therefore, we recommend using it only when Ingress and GatewayAPI don't meet your needs.

GatewayAPI

GatewayAPI↗ 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:



TOC

Via ALB

ALB supports GatewayAPI. Each Gateway resource maps to an ALB resource. Listeners and routes are handled directly by ALB; they are not translated into `Frontend` or `Rule`. Create a GatewayAPI Gateway via ALB

Menu

Guides

Configure Network Policies

Creating Admin Network Policies

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

Configure Cluster Network Policies

Endpoint Health Checker

NodeLocal DNSCache

Creating 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

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:

```yaml
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).

Parameter	Description
	<ul><li>**ExternalName**: Maps the Service to the contents of the externalName field (for example, to the hostname api.foo.bar.example).</li><li>**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.</li></ul>
Target Component	<ul><li>**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` .</li><li>**Virtualization**: The Service will forward requests to a **specific** virtual machine or virtual machine group.</li><li>**Label Selector**: The Service will forward requests to a **certain type** of workload with specified labels, for example, `environment: release` .</li></ul>
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.<ul><li>**Protocol**: The protocol used by the Service, supported protocols include: `TCP` , `UDP` , `HTTP` , `HTTP2` , `HTTPS` , `gRPC` .</li><li>**Service Port**: The service port number exposed by the Service within the cluster, that is, Port, e.g., *80*.</li><li>**Container Port**: The target port number (or name) that the service port maps to, that is, targetPort, e.g., *6379* or *redis*.</li><li>**Service Port Name**: Will be generated automatically. The format is `<protocol>-<service port>-<container port>` , for example: *tcp-*</li></ul>

Parameter	Description
	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   80:30005/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

```
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.	Specifies how traffic reaches the pods.

Key	Value	Description
	• ip: Traffic routes directly to the pods (the cluster must use Amazon VPC CNI).	
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	
kubernetes.io/elb.id	
kubernetes.io/elb.autocreate	Example: `{"type":"public","bandwidth_name":"cce-bandwidt` `1551163379627","bandwidth_chargemode":"bandwidth","bandwic` `["cn-north-4b"],"l4_flavor_name":"L4_flavor.elb.s1.small"}`

Key	
	Note: Please read the Filling Instructions first and adjus
kubernetes.io/elb.subnet-id	
kubernetes.io/elb.class	• union: Shared load balancing. • performance: Exclusive load balancing, only supportec

Key	
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.

Key	Value	Description
loud.google.com/l4-rbs	enabled	Defaults to public. If this parameter is configured, traffic will route directly to the pods.

Creating 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.

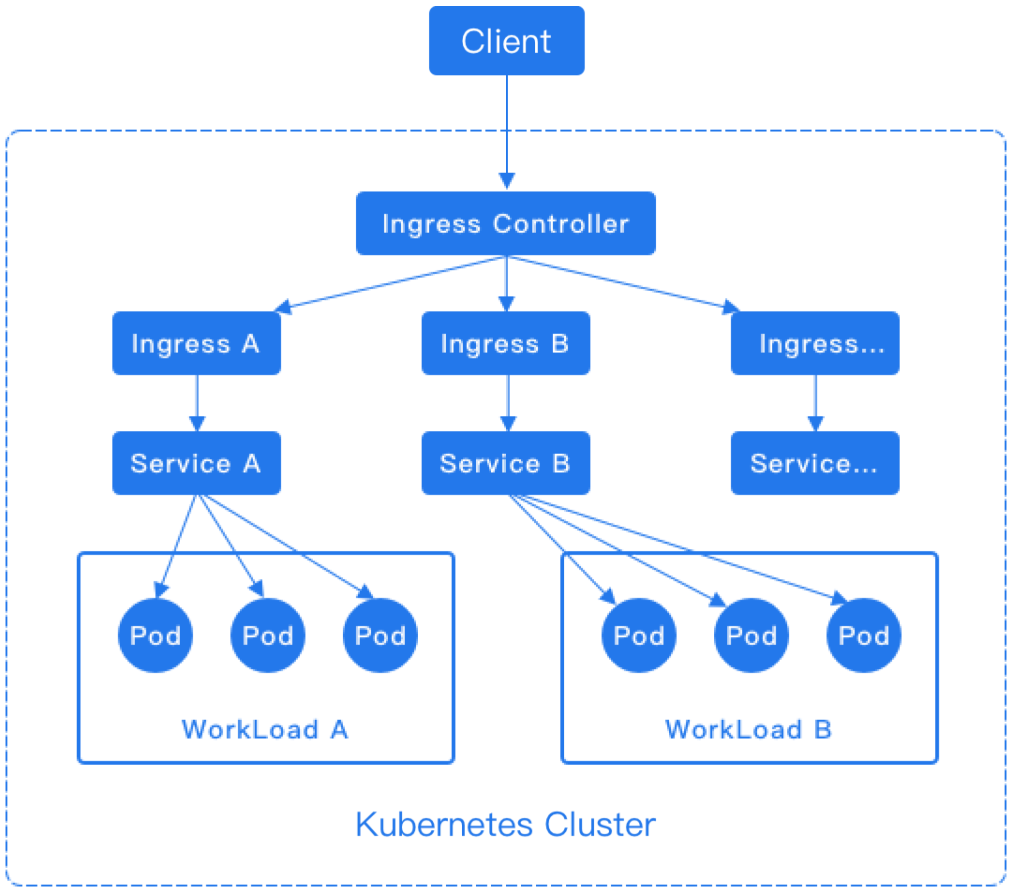
WARNING

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

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
- install alb as ingress-controller

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
```

Creating a Domain Name

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)

```yaml
# test-domain.yaml
apiVersion: crd.alauda.io/v2
kind: Domain
metadata:
  name: "000000000030755752601296 86e67ed4-917a-454a-8553-d55fc4030f81"
  annotations:
    cpaas.io/secret-ref: developer.test.cn-xfd8x 1
  labels:
    cluster.cpaas.io/name: global
    project.cpaas.io/name: cong
spec:
  name: developer.test.cn
  kind: full
```

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
```

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

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**.

Creating External IP Address Pool

An external IP address pool is a collection of IPs that MetalLB utilizes to obtain external access IPs for LoadBalancer type internal routes.

TOC

Prerequisites

If you need to use a BGP type external IP address pool, please contact the administrator to enable the relevant features.

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 MetalLB plugin cannot be successfully deployed.

Deploying the MetalLB Plugin

Using the external address pool relies on the MetalLB plugin.

1. Go to **Administrator**.
2. In the left navigation bar, click **Marketplace** > **Cluster Plugin**.
3. Search MetalLB, click on **MetalLB** to the right of ⋮ > **Deploy**.
4. Wait until the deployment status shows **Deployment Successful** to complete the deployment.

Example IPAddressPool custom resource (CR)

```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
```

1 Ip pool reference.

INFO

Q: What is `L2Advertisement` ?

A:

1. `L2Advertisement` is a Custom Resource (CRD) provided by the MetalLB to control which IP address pool addresses should be broadcast via ARP (IPv4) or NDP (IPv6) in Layer 2 mode.

Q: What is the purpose of `L2Advertisement` ?

A:

1. Specifying which IP addresses in the IPAddressPool to L2 broadcast to (ARP/NDP advertisements);

2. Control broadcast behaviour to prevent IP conflicts or cross-segment broadcasts;
3. Restricting the broadcast range in multi-NIC, multi-network environments.

In short, it tells MetalLB: which IPs can broadcast and to whom (e.g., which nodes).

Without defining a `L2Advertisement` in Layer2 mode, MetalLB will not advertise any addresses.

Q: What is `BGPAdvertisement` in MetalLB?

A:

`BGPAdvertisement` is a Kubernetes Custom Resource Definition (CRD) used in MetalLB ↗, a load-balancer implementation for bare-metal Kubernetes clusters. It controls how IP address ranges (defined in `IPAddressPool`) are advertised to external networks via BGP (Border Gateway Protocol).

Q: Why is `BGPAdvertisement` Important?

A:

In MetalLB's BGP mode, the controller peers with external routers using BGP and advertises the IPs assigned to Kubernetes `Service` objects. The `BGPAdvertisement` resource allows you to:

- Control which address pools are advertised
- Customize route advertisement settings like:
  - Route aggregation
  - BGP communities
  - Local preference (BGP priority)

Without defining a `BGPAdvertisement`, MetalLB will not advertise any addresses, even if you have configured BGP peers.

Creating an External IP Address Pool by using the web console

1. Go to **Administrator**.
2. In the left navigation bar, click **Network Management** > **External IP Address Pool**.

3. Click **Create External IP Address Pool**.

4. Refer to the following instructions to configure certain parameters.

Parameter	Description
Type	• 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`.
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

Parameter	Description
	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**.

Creating an External IP Address Pool by using the CLI

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

View Alarm Policy

1. Go to **Administrator**.
2. In the left navigation bar, click **Network Management** > **External IP Address Pool**.
3. Click **View Alarm Policy** in the upper right corner of the page to view the general alarm policy for MetalLB.

Creating BGP Peers

Nodes that establish connections to exchange routing information either between different AS or within the same AS, which communicate via the BGP protocol.

TOC

Terminology

Term	Explanation
AS Number	AS refers to a collection of routers managed by the same technical administrative organization that use a unified routing policy. Each AS in a BGP network is assigned a unique AS number to distinguish it from different ASs. AS numbers are divided into 2-byte AS numbers and 4-byte AS numbers. • The range of 2-byte AS numbers is 1~65535, where 1~64511 are registered public AS numbers on the Internet, similar to public IP addresses; 64512~65535 are private AS numbers, similar to private IP addresses. • The range of 4-byte AS numbers is 1~4294967295.

Term	Explanation
	Devices that support 4-byte AS numbers can be compatible with devices that support 2-byte AS numbers.

Prerequisites

Please contact the administrator to enable the relevant features.

Example BGPPeer custom resource (CR)

```yaml
# test-bgb-example.yaml
apiVersion: metallb.io/v1beta2
kind: BGPPeer
metadata:
  name: example
  namespace: metallb-system
spec:
  myASN: 64512
  peerASN: 64512
  peerAddress: 172.30.0.3
  peerPort: 180
  nodeSelectors:
    - matchLabels:
        alertmanager: "true"
```

Creating a BGPPeer by using the web console.

1. Go to **Administrator**.

2. In the left navigation bar, click **Network Management** > **BGP Peers**.

3. Click **Create BGP Peer**.

4. Refer to the instructions below to configure the parameters.

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

Creating a BGPPeer by using the CLI

```
kubectl apply -f test-bgb-example.yaml
```

Configure Subnets

TOC

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 **Network Management** > **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.
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.

Parameter	Description
	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
`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

CIDR	blockSize Size	Number of Hosts	Size of a Single IP Pool
`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 **Network Management** > **Subnet**.

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 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)

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

1 Default Network Card Name.

2 Configure Network Card by Node.

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

Procedure:

1. Go to **Administrator**.

2. In the left navigation bar, click **Network Management** > **Bridge Network**.

3. Click **Add Bridge Network**.

4. Configure the relevant parameters based on the following instructions.

   Note:

   - *Target Pod* refers to all Pods scheduled on the current node or Pods in namespaces bound to specific subnets scheduled to the current node. This depends on the scope of the subnet under the bridge network.

   - The nodes in the Underlay subnet must have multiple network cards, and the network card used by the bridge network must be exclusively assigned to the Underlay and cannot carry other traffic, such as SSH. For example, if the bridge network has three nodes planning for eth0, eth0, eth1 for exclusive use by the Underlay, then the default network card can be set as eth0, and the network card for node three can be eth1.

Parameter	Description
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 **Network Management** > **VLAN**.
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 **Network Management** > **Subnet**.
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.

Subnet Management

Updating Gateway by using the web console

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

1. Go to **Administrator**.

2. In the left sidebar, click on **Network Management** > **Subnets**.

3. Click the name of the subnet.

4. Select **Action** > **Update Gateway**.

5. Update the parameter configurations; refer to the Parameter Description for details.

6. Click **OK**.

Updating Gateway by using the CLI

```
kubectl patch subnet test-overlay-subnet --type=json -p='[
  {"op": "replace", "path": "/spec/gatewayType", "value": "centralized"},
  {"op": "replace", "path": "/spec/gatewayNode", "value": "192.168.66.210"},
  {"op": "replace", "path": "/spec/natOutgoing", "value": true},
  {"op": "replace", "path": "/spec/enableEcmp", "value": true}
]'
```

Updating Reserved IPs by using the web console

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

1. Go to **Administrator**.

2. In the left sidebar, click on **Network Management** > **Subnets**.

3. Click the name of the subnet.

4. Select **Action** > **Update Reserved IP**.

5. After completing the updates, click **Update**.

Updating Reserved IPs by using the CLI

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

Assigning Projects by using the web console

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

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

Assigning Projects by using the CLI

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

Assigning Namespaces by using the web console

Assigning subnets to specific namespaces allows for finer network isolation.

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

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

Assigning Namespaces by using the CLI

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

Expanding Subnets by using the web console

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

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

Expanding Subnets by using the CLI

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

Managing Calico Networks

Support for assigning projects and namespaces; for details, please refer to the project assignment and namespace assignment.

Delete Subnet by using the web console

NOTE

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

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

Delete Subnet by using the CLI

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

Creating Network Policies

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

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

Area	Parameter		Description
			`spec.policyTypes` fields are 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.

Area	Parameter		Description
		IP Range	Enter the CIDR that can access the target Pod and can exclude CIDR ranges that are not allowed 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.

1. 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 ↗.

Creating Admin Network Policies

INFO

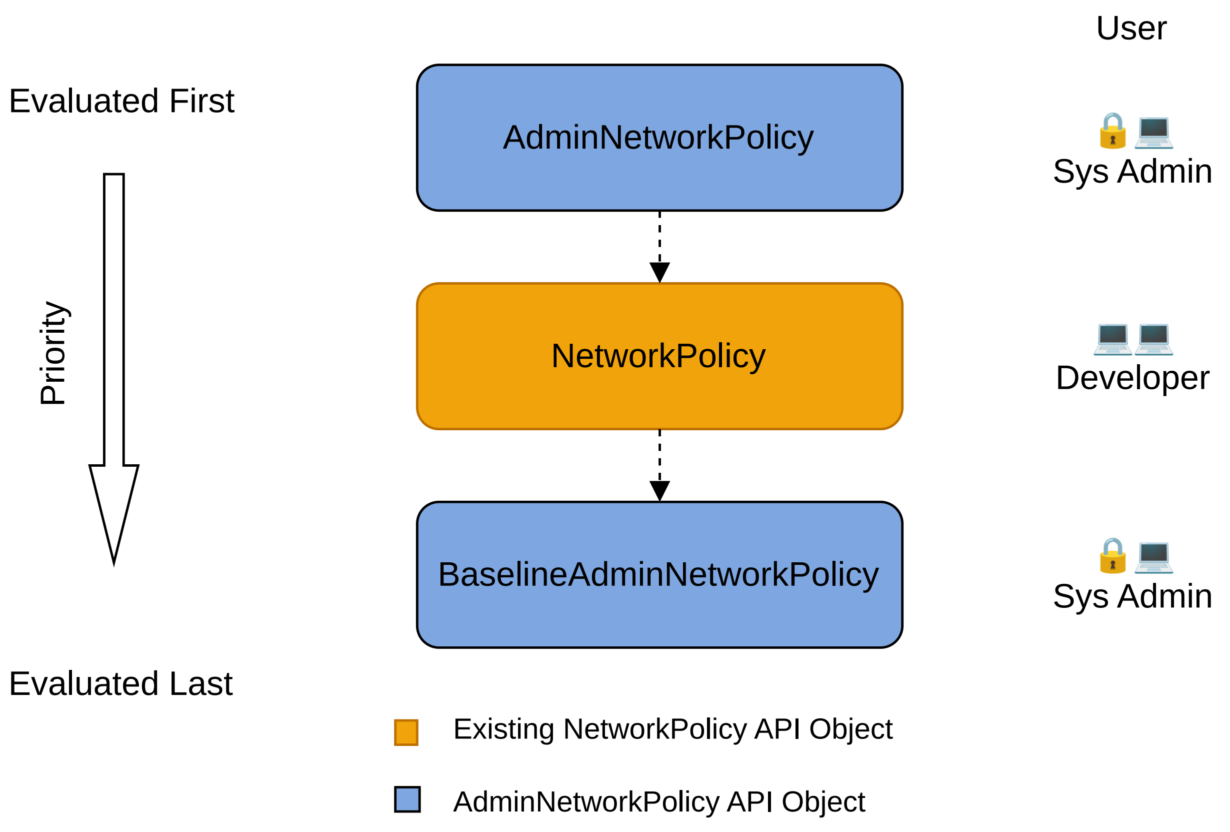
The platform now provides two different UIs for Cluster Network Policies. The old one is maintained for compatibility reasons, while the new one is more flexible and provides a native YAML editor. We recommend using the new version.

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

The new cluster network policy adopts the Kubernetes community's Admin Network Policy ↗ standard design, providing more flexible configuration methods and rich configuration options.

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：



TOC

Notes

- Only Kube-OVN CNI supports admin network policies.
- In Kube-OVN network mode, this feature is at Alpha maturity level.
- 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 **Network** > **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 logical **OR**	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 the Pods affected by the current rule.

Area	Parameter		Description
	relationship between them.		<ul><li>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.</li><li>If this item is not configured, all Pods in all namespaces in the cluster can access the target Pod by default.</li></ul>
		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.
	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.
Egress	Rule	Node Selector	Specifies which node IPs the target Pods are allowed to

Area	Parameter		Description
	Description: If multiple sources are added in the rule, there is a logical **OR** relationship between them.		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
```

Additional resource

- Configure Cluster Network Policies

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

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

TOC

Installing Multus CNI

Deploying the Multus CNI Plugin

1. Go to **Administrator**.
2. In the left navigation bar, click **Marketplace** > **Cluster Plugins**.
3. In the search bar, type "multus" to find the Multus CNI plugin.

4. Locate the **"Alauda Container Platform Networking for Multus"** plugin in the list.
5. Click the three dots (⋮) next to the plugin entry and select **Install**.
6. The plugin will be deployed to your cluster. You can monitor the installation status in the **State** column.

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

Creating Subnets

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

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

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

After creation, apply the resource:

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

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

NOTE

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

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

After creation, apply the resource:

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

Creating Pod with Multiple Network Interfaces

Create a pod according to the following example.

NOTE

- The `metadata.annotations` must contain a key-value pair `k8s.v1.cni.cncf.io/networks=default/attachnet`, where the value format is `<NAMESPACE>/<NAME>`, and `<NAMESPACE>` and `<NAME>` are the namespace and name of the NetworkAttachmentDefinition CR respectively.
- If the Pod needs three network interfaces, configure the value of `k8s.v1.cni.cncf.io/networks` as `default/attachnet,default/attachnet2`.

```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
```

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

Verifying Dual Network Interface Creation

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

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

Additional Features

Fixed IP

- **Primary Network Interface (First Interface)**: If you need to fix the IP of the primary network interface, the method is the same as using a fixed IP with a single network interface. Add the annotation `ovn.kubernetes.io/ip_address=<IP>` to the Pod.

- **Secondary Network Interface (Second Interface or Other Interfaces)**: The basic method is similar to the primary network interface, with the difference that the `ovn` in the Annotation Key is replaced with the corresponding NetworkAttachmentDefinition provider. Example: `attachnet.default.ovn.kubernetes.io/ip_address=172.170.0.101`.

Additional Routes

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

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

Configure Cluster Network Policies

Cluster network policies are responsible for managing **project-level** access control rules. When this feature is enabled, different projects are isolated from each other by default, and compute components in different projects cannot access each other over the network. Communication can be achieved by adding **Access between Projects** or **Access Using IP Addresses** rules.

Once configured, the cluster network policies will be synchronized to the namespaces under the cluster, and can be viewed in the **Network Policies** feature module of the container platform.

TOC

Notes

- The effectiveness of the cluster network policies depends on whether the network plugin used by the cluster supports network policies.
  - Kube-OVN and Calico support network policies.
  - Flannel does not support network policies.
  - When accessing the cluster or using a custom network plugin, you can refer to the relevant documentation to confirm support.
- The functionality is in Alpha maturity under the Kube-OVN network mode.

Procedure

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

Configuration Item	Description
Complete Isolation Between Projects	Whether to enable the complete isolation switch between projects, which is enabled by default and can be turned off by clicking. When enabled, network isolation is achieved between all projects in the current cluster, and other resources are not allowed to access any project within the cluster (e.g., external IPs, load balancers). This does not affect projects' access to resources outside the cluster.
Single Project Access	This parameter is only effective when the **Complete Isolation Between Projects** switch is enabled. Configure the **source project** and **target project** for one-way access. Click **Add** to add a configuration record, supporting multiple records. In the **source project** dropdown, select a project that will access the target project or select all projects; in the **target project** dropdown, select the target project to be accessed.
IP Segment Access	This parameter is only effective when the **Complete Isolation Between Projects** switch is enabled. Configure the specific **IP/segment** and **target project** for one-way access. Click **Add** to add a configuration record, supporting multiple records. In the **source IP segment** input box, enter the IP or CIDR

Configuration Item	Description
	segment to access the target project; in the **target project** dropdown, select the target project to be accessed.

5. Click **Configure**.

Configure Egress Gateway

TOC

About Egress Gateway

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

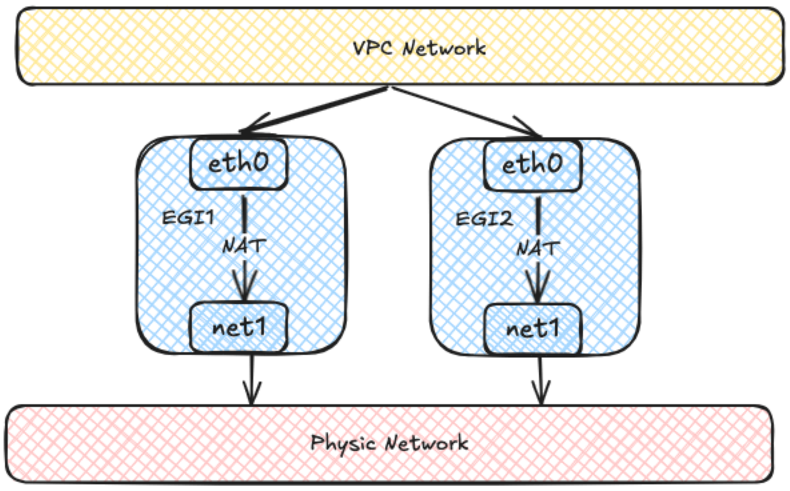
- Achieves Active-Active high availability through ECMP, enabling horizontal throughput scaling
- Implements fast failover (<1s) via BFD
- Supports IPv6 and dual-stack
- Enables granular routing control through NamespaceSelector and PodSelector
- Allows flexible scheduling of Egress Gateway through NodeSelector

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

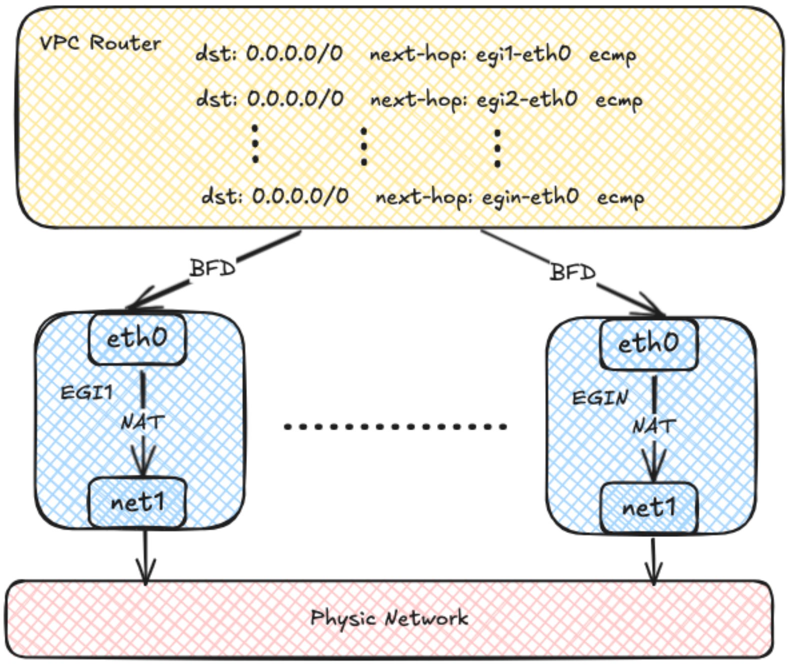
- Uses macvlan for underlying network connectivity, requiring Underlay support from the underlying network
- In multi-instance Gateway mode, multiple Egress IPs are required
- Currently, only supports SNAT; EIP and DNAT are not supported
- Currently, recording source address translation relationships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

Implementation Details

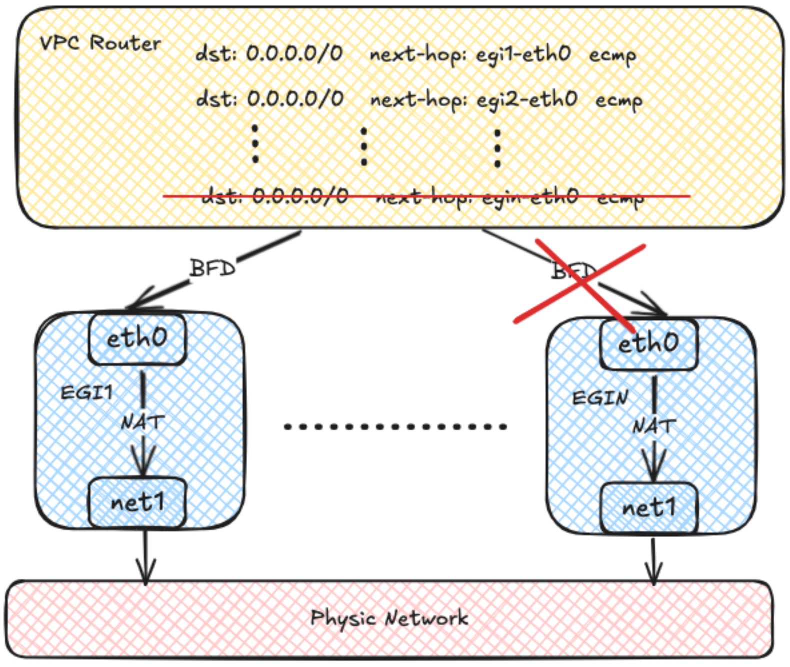
Each Egress Gateway consists of multiple Pods with multiple network interfaces. Each Pod has two network interfaces: one joins the virtual network for communication within the VPC, and the other connects to the underlying physical network via Macvlan for external network communication. Virtual network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



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


VPC Router
dst: 0.0.0.0/0 next-hop: egi1-eth0 ecmp
dst: 0.0.0.0/0 next-hop: egi2-eth0 ecmp
dst: 0.0.0.0/0 next-hop: egin-eth0 ecmp
BFD
BFD
eth0
EGI1
NAT
net1
eth0
EGIN
NAT
net1
Physic Network

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


VPC Router
dst: 0.0.0.0/0 next-hop: egi1-eth0 ecmp
dst: 0.0.0.0/0 next-hop: egi2-eth0 ecmp
~~dst: 0.0.0.0/0 next-hop: egin-eth0 ecmp~~
BFD
BFD
eth0
EGI1
NAT
net1
eth0
EGIN
NAT
net1
Physic Network

Notes

- Only Kube-OVN CNI supports Egress Gateway.
- Egress Gateway requires Multus-CNI.

Usage

Creating a Network Attachment Definition

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

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

1 Host interface that connects to the external network.

2 Provider name with a format of `<network attachment definition name>.<namespace>`.

3 Provider name used to identify the external network and MUST be consistent with the one in the NetworkAttachmentDefinition.

TIP

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

Creating a VPC Egress Gateway

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

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

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

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

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

NOTICE

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

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

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

To view more informations:

```
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         macvlan1          Completed   true    ["10.16.0.12"]   ["172.17.0.11"]   ["kube-ovn-
worker"]   82s
```

To view the workload:

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

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

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

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

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

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

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

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

Capture packets in the Gateway Pod to verify network traffic:

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

Routing policies are automatically created on the OVN Logical Router:

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

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

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

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

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

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

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

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

Enabling BFD-based High Availability

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

```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 is running in Active-Backup mode.

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

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

1 BFD Port created on the OVN Logical Router.

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

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

1 VPC to which the Egress Gateway belongs.

2 Internal subnet to which the Egress Gateway instances are connected.

3 External subnet to which the Egress Gateway instances are connected.

4 Whether to enable BFD for the Egress Gateway.

5 Minimum receive interval for BFD, in milliseconds.

6 Minimum transmit interval for BFD, in milliseconds.

7 Multiplier for BFD, which determines the number of missed packets before declaring a failure.

To view VPC Egress Gateway information:

```
$ kubectl get veg gateway2 -o wide
NAME       VPC    REPLICAS   BFD ENABLED   EXTERNAL SUBNET   PHASE       READY   INTERNAL IPS                    EXTERNAL IPS
WORKING NODES                            AGE
gateway2   vpc1   2          true          macvlan           Completed   true    ["10.16.0.102","10.16.0.103"]
["172.17.0.13","172.17.0.14"]   ["kube-ovn-worker","kube-ovn-worker2"]   58s

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

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

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

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

To view BFD connections:

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

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

NOTICE

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

Configuration Parameters

VPC BFD Port

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

VPC Egress Gateway

Fields	Type	Optional	Default Value	Descripti
vpc	string	Yes	Name of the default VPC	VPC name.

Fields			Type	Optional	Default Value	Descripti	
					(ovn-cluster)		
replicas			integer/int32	Yes	1	Replicas.	
prefix			string	Yes	-	Immutable prefix of the workload	
image			string	Yes	-	The image used by the workload	
internalSubnet			string	Yes	Name of the default subnet within the VPC.	Name of the subnet used to acce network.	
externalSubnet			string	No	-	Name of the subnet used to acce network.	
internalIPs			string array	Yes	-	IP addresses used for accessing network. IPv4, IPv6 and dual-stac The number of IPs specified must *replicas*. It is recommended to set the num avoid extreme cases where the P properly.	
externalIPs			string array	Yes	-	IP addresses used for accessing network. IPv4, IPv6 and dual-stac The number of IPs specified must *replicas*. It is recommended to set the num avoid extreme cases where the P properly.	
bfd	enabled		boolean	Yes	false	BFD Configuration.	Wheth Egres
bfd	minRX		integer/int32	Yes	1000	BFD Configuration.	BFD m millise
bfd	minTX		integer/int32	Yes	1000	BFD Configuration.	BFD m millise
bfd	multiplier		integer/int32	Yes	3	BFD Configuration.	BFD m
policies	snat		boolean	Yes	false	Egress policies.	Wheth SNAT/
policies	ipBlocks		string array	Yes	-	Egress policies.	IP ran gatew Both I suppo
policies	subnets		string array	Yes	-	Egress policies.	The V the ga IPv4, I subne
selectors	namespaceSelector	matchLabels	object	Yes	-	Configure Egress policies by namespace selectors and Pod selectors.	Name select empty select

Fields			Type	Optional	Default Value	Descripti	
		matchExpressions	object array	Yes	-	SNAT/MASQUERADE will be applied to the matched Pods.	match names
	podSelector	matchLabels	object	Yes	-		Pod s An em label s match Pods.
		matchExpressions	object array	Yes	-		
nodeSelector	matchLabels		object	Yes	-	Node selector used to select nodes that carries the workload deployment. The workload (Deployment/Pod) will run on the selected nodes.	A map
	matchExpressions		object array	Yes	-		A list c requir requir
	matchFields		object array	Yes	-		A list c requir requir
trafficPolicy			string	Yes	Cluster	**Effective only when BFD is ena** Available values: *Cluster*/*Local*. When set to *Local*, Egress traffic VPC Egress Gateway instance ru if available. If the instance is down, Egress tra other instances.	

Additional resources

- Egress Gateway - Kube-OVN Document ↗
- RFC 5880 - Bidirectional Forwarding Detection (BFD) ↗

Network Observability

TOC

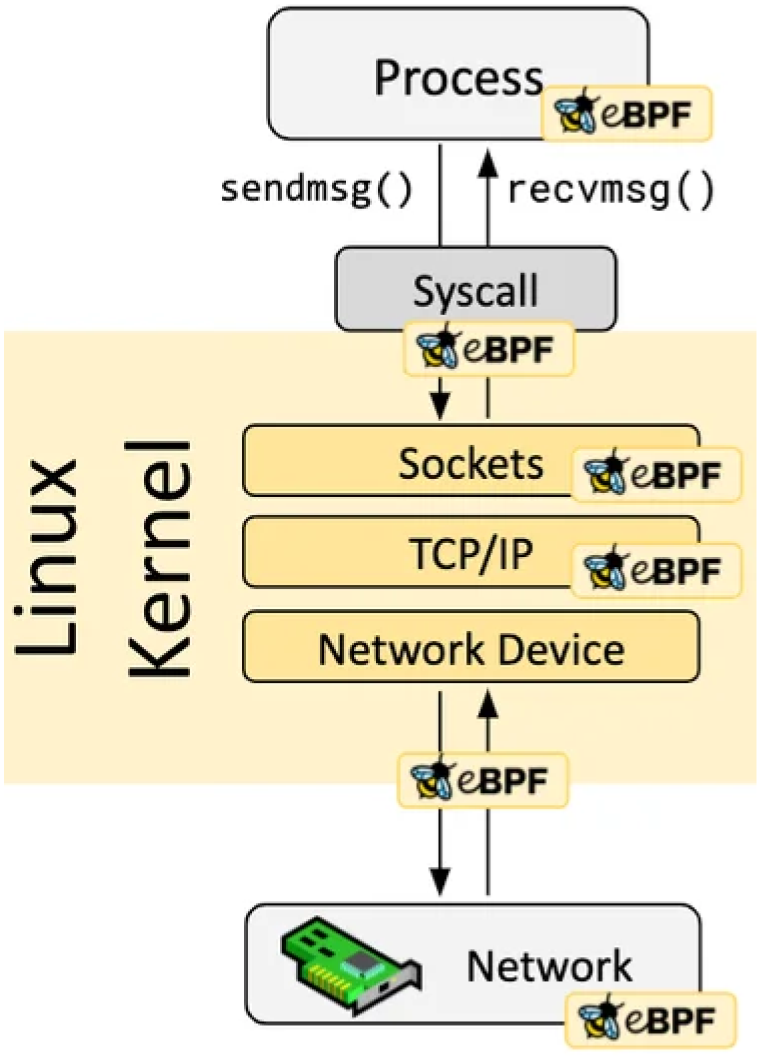
About DeepFlow

What is DeepFlow

The DeepFlow open-source project aims to provide deep observability for complex cloud-native and AI applications. DeepFlow implemented Zero Code data collection with eBPF for metrics, distributed tracing, request logs and function profiling, and is further integrated with SmartEncoding to achieve Full Stack correlation and efficient access to all observability data. With DeepFlow, cloud-native and AI applications automatically gain deep observability, removing the heavy burden of developers continually instrumenting code and providing monitoring and diagnostic capabilities covering everything from code to infrastructure for DevOps/SRE teams.

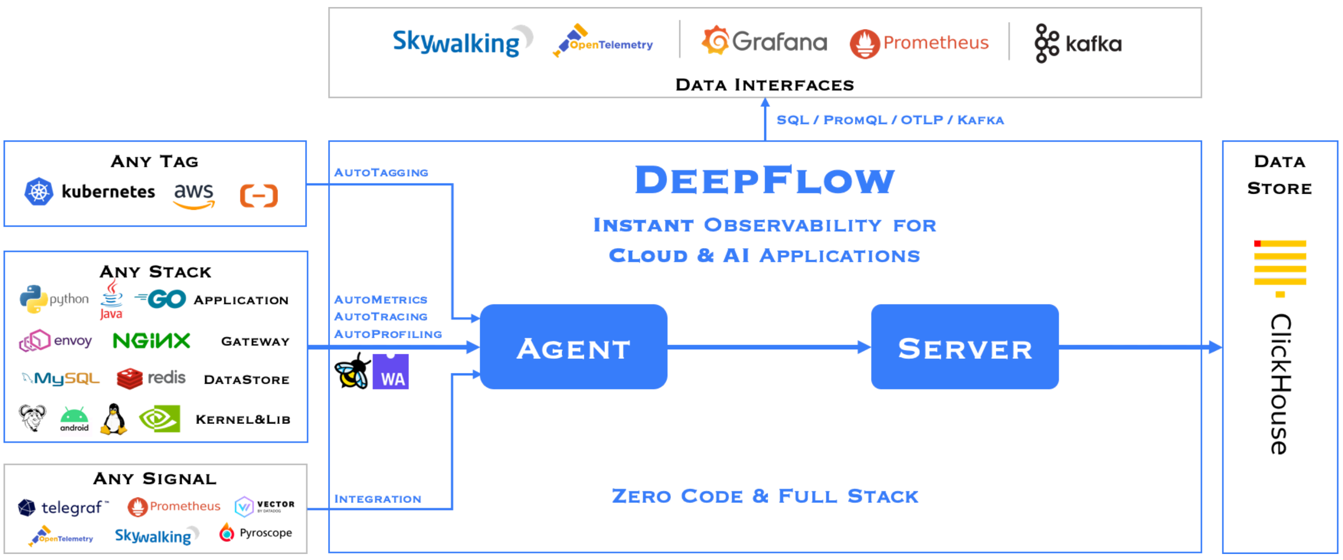
Using eBPF Technology

Assuming you have a basic understanding of eBPF, it is a secure and efficient technology for extending kernel functionality by running programs in a sandbox, a revolutionary innovation compared to traditional methods of modifying kernel source code and writing kernel modules. eBPF programs are event-driven, and when the kernel or user programs pass through an eBPF Hook, the corresponding eBPF program loaded at the Hook point will be executed. The Linux kernel predefines a series of commonly used Hook points, and you can also dynamically add custom Hook points in the kernel and applications using kprobe and uprobe technologies. Thanks to Just-in-Time (JIT) technology, the execution efficiency of eBPF code can be comparable to native kernel code and kernel modules. Thanks to the Verification mechanism, eBPF code will run safely without causing kernel crashes or entering infinite loops.



Software Architecture

DeepFlow consists of two components, Agent and Server. An Agent runs in each K8s node, legacy host and cloud host, and is responsible for AutoMetrics and AutoTracing data collection of all application processes on the host. Server runs in a K8s cluster and provides Agent management, tag injection, data ingest and query services.



Install DeepFlow

Introduction

Kernel Requirements

The eBPF capabilities (AutoTracing, AutoProfiling) in DeepFlow have the following kernel version requirements:

Architecture	Distribution	Kernel Version	kprobe	Golang uprobe	OpenSSL uprobe	perf
X86	CentOS 7.9	3.10.0 [1]	Y	Y [2]	Y [2]	Y
	RedHat 7.6	3.10.0 [1]	Y	Y [2]	Y [2]	Y
	*	4.9-4.13				Y
		4.14 [3]	Y	Y [2]		Y
		4.15	Y	Y [2]		Y
		4.16	Y	Y		Y
		4.17+	Y	Y	Y	Y
ARM	CentOS 8	4.18	Y	Y	Y	Y
	EulerOS	5.10+	Y	Y	Y	Y
	KylinOS V10 SP2	4.19.90-25.24+	Y	Y	Y	Y
	KylinOS V10 SP3	4.19.90-52.24+	Y	Y	Y	Y
	Other Distributions	5.8+	Y	Y	Y	Y

Additional notes on kernel versions:

1 CentOS 7.9 and RedHat 7.6 have backported some eBPF capabilities (opens new window)into the 3.10 kernel. In these two distributions, the detailed kernel versions supported by DeepFlow are as follows (dependent hook points):

- 3.10.0-957.el7.x86_64
- 3.10.0-1062.el7.x86_64
- 3.10.0-1127.el7.x86_64
- 3.10.0-1160.el7.x86_64

2 Golang/OpenSSL processes inside containers are not supported.

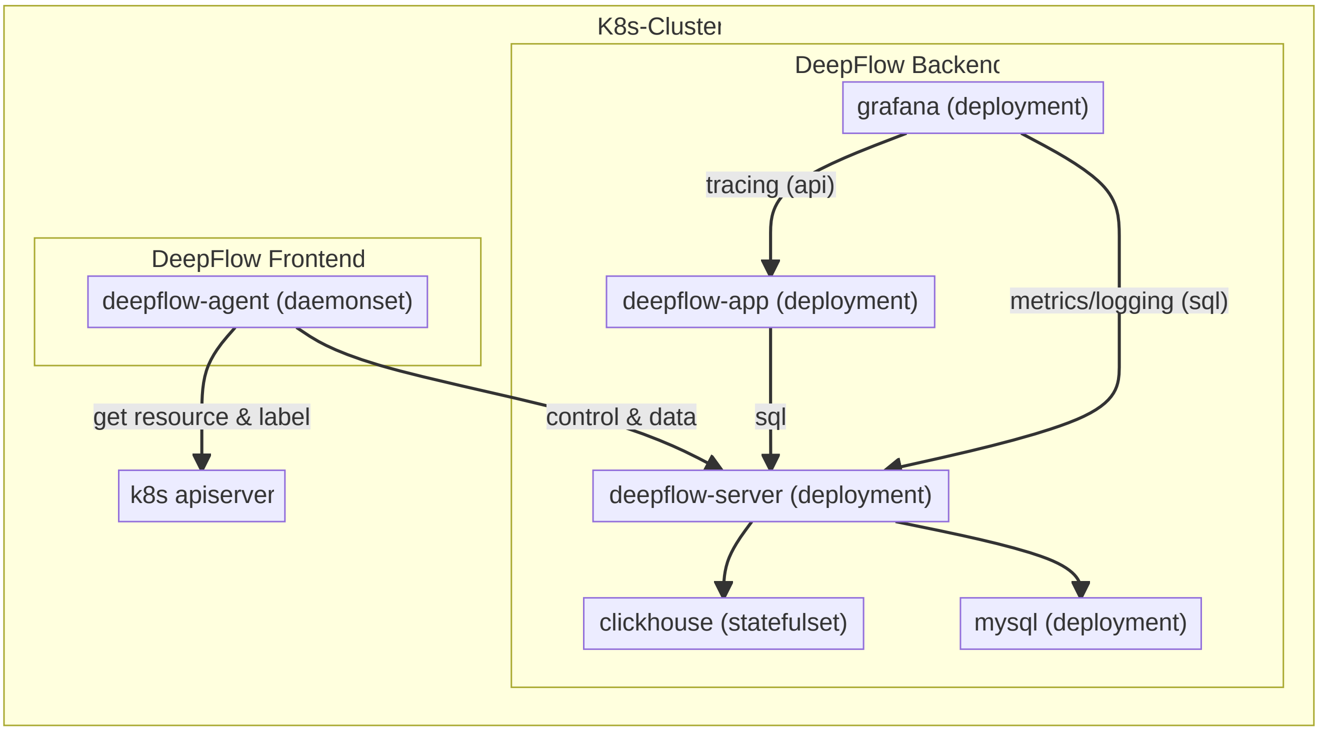
3 In kernel version 4.14, a tracepoint cannot be attached by multiple eBPF programs (e.g., two or more deepflow-agents cannot run simultaneously), this issue does not exist in other versions

NOTE

RedHat's statement:

> The eBPF in Red Hat Enterprise Linux 7.6 is provided as Tech Preview and thus doesn't come with full support and is not suitable for deployment in production. It is provided with the primary goal to gain wider exposure, and potentially move to full support in the future. eBPF in Red Hat Enterprise Linux 7.6 is enabled only for tracing purposes, which allows attaching eBPF programs to probes, tracepoints and perf events.

Deployment Topology



Preparation

Storage Class

We recommend using Persistent Volumes to store MySQL and ClickHouse data to avoid unnecessary maintenance costs. You can provide a default Storage Class or add the *--set global.storageClass=<your storageClass>* parameter to select a Storage Class for creating PVC.

For more information on storage configuration, please refer to the Storage documentation.

Package

Download the DeepFlow package

Visit the Custom Portal to download the DeepFlow package.

If you don't have access to the Custom Portal, contact technical support.

Upload the package to the platform

Use the violet tool to publish the package to the platform.

For detailed instructions on using this tool, refer to the CLI.

Install

Create an application resource in the *cpaas-system* namespace to deploy DeepFlow. Here is an example manifest:

```
apiVersion: app.k8s.io/v1beta1
kind: Application
metadata:
  name: deepflow
  namespace: cpaas-system
  annotations:
    app.cpaas.io/chart.source: public-charts/deepflow-plugin 1
    app.cpaas.io/chart.version: v4.1.0 2
    app.cpaas.io/chart.values: |
      {
        "global": {
          "storageClass": "example-sc" 3
        },
        "server": {
          "service": {
            "type": "ClusterIP"
          }
        },
        "deepflow-agent": {
          "clusterNAME": "cluster1" 4
        },
        "grafana": {
          "adminUser": "admin", 5
          "adminPassword": "password", 6
          "service": {
            "type": "ClusterIP"
          },
          "grafana.ini": {
            "server": {
              "root_url": "%(protocol)s://%(domain)s/clusters/cluster1/deepflow", 7
              "serve_from_sub_path": true
            }
          }
        },
        "mysql": {
          "storageConfig": {
            "persistence": {
              "size": "50G" 8
            }
          }
        },
        "clickhouse": {
          "storageConfig": {
            "persistence": [
              {
                "accessModes": [
                  "ReadWriteOnce"
                ],
                "name": "clickhouse-path",
                "size": "100Gi", 9
                "storageClass": "{{ .Values.global.storageClass }}"
              },
              {
                "accessModes": [
                  "ReadWriteOnce"
                ],
                "name": "clickhouse-storage-path",
                "size": "200Gi", 10
                "storageClass": "{{ .Values.global.storageClass }}"
              }
            ]
          }
        }
      }
    cpaas.io/display-name: 'DeepFlow'
  labels:
    sync-from-helmrequest: 'true'
```

1 Source of the DeepFlow chart. If you want to use a different chart, please contact technical support.

2 Version of the DeepFlow chart. MUST match the version of the DeepFlow package uploaded to the platform.

3 Storage Class used to create Persistent Volumes for MySQL and ClickHouse.

4 Name of the cluster where DeepFlow is deployed.

5 Username of the Grafana web UI. You can change it to your desired username.

6 Password of the Grafana web UI. It's recommended to change it to a secure password.

7 The root URL for Grafana web UI. You can change it to your desired URL.

8 Size of the persistent volume for MySQL. Please adjust it according to your needs.

9 Size of the persistent volume for ClickHouse path. Please adjust it according to your needs.

10 Size of the persistent volume for ClickHouse storage path. Please adjust it according to your needs.

Wait for the application to be ready:

```
kubectl -n cpaas-system wait --for=jsonpath='{.spec.assemblyPhase}'=Succeeded application
deepflow
kubectl -n cpaas-system rollout status statefulset deepflow-clickhouse
kubectl -n cpaas-system rollout status deployment deepflow-mysql
kubectl -n cpaas-system rollout status deployment deepflow-server
kubectl -n cpaas-system rollout status deployment deepflow-app
kubectl -n cpaas-system rollout status deployment deepflow-grafana
kubectl -n cpaas-system rollout status daemonset deepflow-agent
```

Configure Ingress for Grafana web UI

Create an Ingress resource to access the Grafana web UI. Here is an example manifest:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    nginx.ingress.kubernetes.io/backend-protocol: http
    nginx.ingress.kubernetes.io/enable-cors: "true"
  name: deepflow-grafana
  namespace: cpaas-system
spec:
  ingressClassName: cpaas-system 1
  rules:
  - http:
      paths:
      - backend:
          service:
            name: deepflow-grafana
            port:
              number: 80
        path: /clusters/${CLUSTER_NAME}/deepflow($|/)(.*) 2
        pathType: ImplementationSpecific
```

1 Ingress Class name. If you are installing DeepFlow in the global cluster, set it to *global-alb2*. You can also set it to your custom Ingress Class name.

2 This path MUST match the root URL configured in the Application resource.

Access the Grafana web UI

You can access the Grafana web UI via the URL specified in the Ingress resource, and login with the username and password you set in the Application resource.

> **NOTICE**
>
> It's highly recommended to change the password after the first login.

Additional resources

- Instant Observability for Cloud & AI Applications
- eBPF - Introduction, Tutorials & Community Resources

Configure ALB Rules

TOC

Introduction

What is a 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

install alb and ft

Quick demo of rule

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

```yaml
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

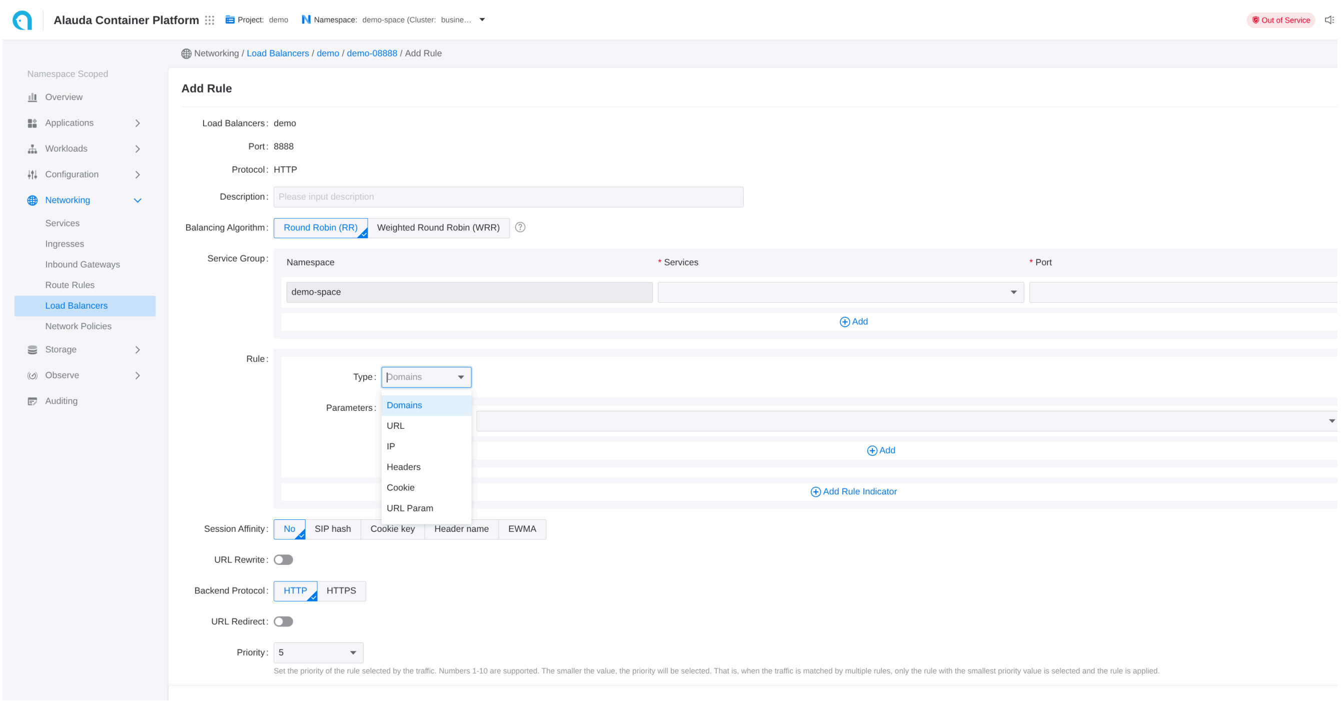
You can configure one or more services within a rule.

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.

Creating Rule

Using web console



Networking / Load Balancers / demo / demo-08888 / Add Rule

Add Rule

Load Balancers: demo

Port: 8888

Protocol: HTTP

Description: Please input description

Balancing Algorithm: [✓ Round Robin (RR)] Weighted Round Robin (WRR)

Service Group:

Namespace	* Services	* Port
demo-space		

Add

Rule:

Type: Domains

Parameters:

Domains
URL
IP
Headers
Cookie
URL Param

Add

Add Rule Indicator

Session Affinity: [✓ No] SIP hash Cookie key Header name EWMA

URL Rewrite:

Backend Protocol: [✓ HTTP] HTTPS

URL Redirect:

Priority: 5

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

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

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

using the CLI

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

Https

If the frontend protocol (ft) is HTTPS or GRPCS, the rule can also be configured to use HTTPS. You can specify the certificate either in the rule or in the ingress to match the certificate for that specific port.

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

Certificate Annotation in Ingress

Certificates can be referenced across namespaces via annotation.

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

Certificate in Rule

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

TLS Mode

Edge Mode

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

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

Re-encrypt Mode

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

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

Ingress

ingress sync

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

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

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

NOTE

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

2.1. `spec.source.type` currently only supports `ingress`.

2.2. `spec.source.name` is ingress name.

2.3. `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

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

- 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 afterwards.
- 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), Docker deployment, and Containerd deployment.

Deploy Deployment

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

Operation Steps

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

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

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

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

   Modified parameter configurations are as follows:

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

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

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

Docker 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 Docker deployment**

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

     - Command for the Leader node:

```
docker 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="192.168.39.22,192.168.39.24,192.168.39.37"
<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:

```
docker 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="192.168.39.22,192.168.39.24,192.168.39.37"
<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 Containerd deployment**

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

- Command for the Leader node:

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

- Commands for the other two nodes:

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

Deploy the cluster interconnection controller in the Global cluster

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

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

```
cat << EOF |kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  name: ovn-ic
  namespace: cpaas-system
data:
  ic-db-host: "192.168.39.22,192.168.39.24,192.168.39.37"   # Address of the node where
the cluster interconnect controller is located, in this 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.
2. Click the name of the ***cluster*** that you want to decommission. 3.
3. In the upper-right corner, click **Options** > **Cluster Interconnect**. 4.
4. Click **Exit cluster interconnection** for the cluster you want to exit. 5. Enter the cluster name correctly.
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 xxxxxx"}
```

Operational Steps

1. Exit the interconnected cluster for the cluster to be joined.
2. Execute the cleanup script in the container or pod.

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

   Method 1: Execute in ovn-ic-db container

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

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

Then execute one of the following cleanup commands:

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

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

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

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

Method 2: Execute in ovn-ic-controller pod

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

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

  Then execute one of the following cleanup commands:

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

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

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

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

Uninstalling the Interconnected Cluster

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

Operational Steps

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

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

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

   - Exit the interconnected cluster through platform operations.

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

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

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

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

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

   - Docker command:

     ```
     docker stop ovn-ic-db
     docker 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: docker
  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 docker-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.
docker-ts-71292a21-131d-492a-9f0c-0611af458950 100
docker-ts-1de7ee15-f372-4ab9-8c85-e54d61ea18f1 99
```

Endpoint Health Checker

TOC

Overview

The Endpoint Health Checker is a cluster plugin designed to monitor and manage the health status of service endpoints. It automatically removes unhealthy endpoints from load balancers 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
- **Load Balancer Integration**: Automatically removes unhealthy endpoints from load balancer rotation
- **Service Availability**: Ensures traffic is only directed to healthy, available endpoints

---

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. 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
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
   - **HTTP/HTTPS response validation**: Sends HTTP requests and validates response codes and content

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 or HTTP 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 load balancer rotation

Activation Methods

Health checking can be activated through two methods:

1. **Pod-level annotation (Recommended)**:

   Add the annotation to the pod template in your Deployment:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-demo
spec:
  replicas: 10
  selector:
    matchLabels:
      app: nginx-demo
  template:
    metadata:
      labels:
        app: nginx-demo
      annotations:
        endpoint-health-checker.io/enabled: "true"
    spec:
      containers:
        - name: nginx
          image: nginx:alpine
          ports:
            - containerPort: 80
          livenessProbe:
            tcpSocket:
              port: 80
            initialDelaySeconds: 15
            periodSeconds: 10
          readinessProbe:
            tcpSocket:
              port: 80
            initialDelaySeconds: 5
            periodSeconds: 5
```

2. **Pod-level readinessGates (Legacy)**:

   Configure readinessGates in the pod spec for older versions:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-demo-legacy
spec:
  replicas: 5
  selector:
    matchLabels:
      app: nginx-demo-legacy
  template:
    metadata:
      labels:
        app: nginx-demo-legacy
    spec:
      readinessGates:
        - conditionType: "endpointHealthCheckSuccess"
      containers:
        - name: nginx
          image: nginx:alpine
          ports:
            - containerPort: 80
          livenessProbe:
            tcpSocket:
              port: 80
            initialDelaySeconds: 15
            periodSeconds: 10
          readinessProbe:
            tcpSocket:
              port: 80
            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.

NodeLocal DNSCache

TOC

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.

Installation

Install via Marketplace

1. Navigate to **Administrator** > **Marketplace** > **Cluster Plugins**.
2. Search for "**Alauda Build of NodeLocal DNSCache**" 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
```

Menu

How To

Preparing Kube-OVN Underlay Physical Network

Soft Data Center LB Solution (Alpha)

Automatic Interconnection of Underlay and Overlay Subnets

Install Ingress-Nginx via Cluster Plugin

Install Ingress-Nginx via Cluster Plugin

Tasks for Ingress-Nginx

Tasks for Ingress-Nginx

ALB

Configure GatewayApi Gateway via ALB

Bind NIC in ALB

Decision-Making for ALB Performance Selection

Calico Network Supports WireGuard Encryption

Kube-OVN Overlay Network Supports IPsec Encryption

Kube-OVN Overlay Network Supports IPsec Encryption

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.

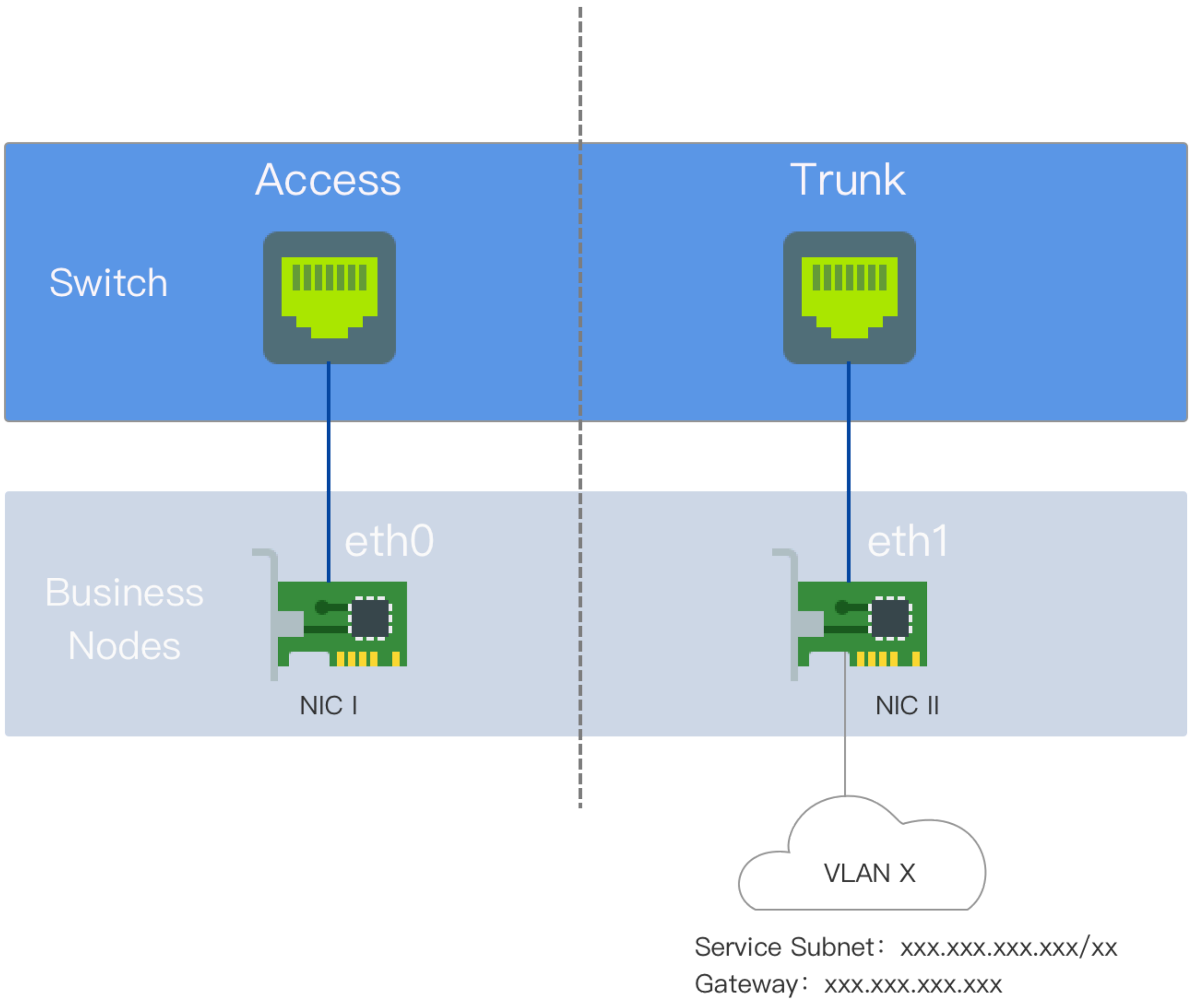
TOC

Usage Instructions

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

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

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



Terminology Explanation

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

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

The main advantages of VLANs include:

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

Environment Requirements

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

- **Network Configuration Requirements**
  - Kube-OVN checks the gateway's connectivity via ICMP protocol when starting containers; the underlying gateway must respond to ICMP requests.
  - For service access traffic, Pods will first send packets to the gateway, which must have the ability to forward packets back to the local subnet.
  - When the switch or bridge has Hairpin functionality enabled, **Hairpin must be disabled**. If using a VMware virtual machine environment, set **Net.ReversePathFwdCheckPromisc** on the VMware host to **1**, and Hairpin does not need to be disabled.

- The bridging NIC **cannot** be a **Linux Bridge**.
- NIC bonding modes support Mode 0 (balance-rr), Mode 1 (active-backup), Mode 4 (802.3ad), Mode 6 (balance-alb), with a recommendation to use 0 or 1. Other bonding modes have not been tested; please use them with caution.

- **IaaS (Virtualization) Layer Configuration Requirements**
  - For OpenStack VM environments, the **PortSecurity** for the corresponding network port needs to be disabled.
  - For VMware's vSwitch network, **MAC Address Changes**, **Forged Transmits**, and **Promiscuous Mode Operation** must all be set to **Accept**.
  - For public clouds such as AWS, GCE, and Alibaba Cloud, Underlay mode networks cannot be supported due to their lack of user-defined MAC address capabilities.

Configuration Example

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

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

Switch Configuration

Configure the VLAN Interface:

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

Configure the interface connected to NIC Two:

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

Check Network Connectivity

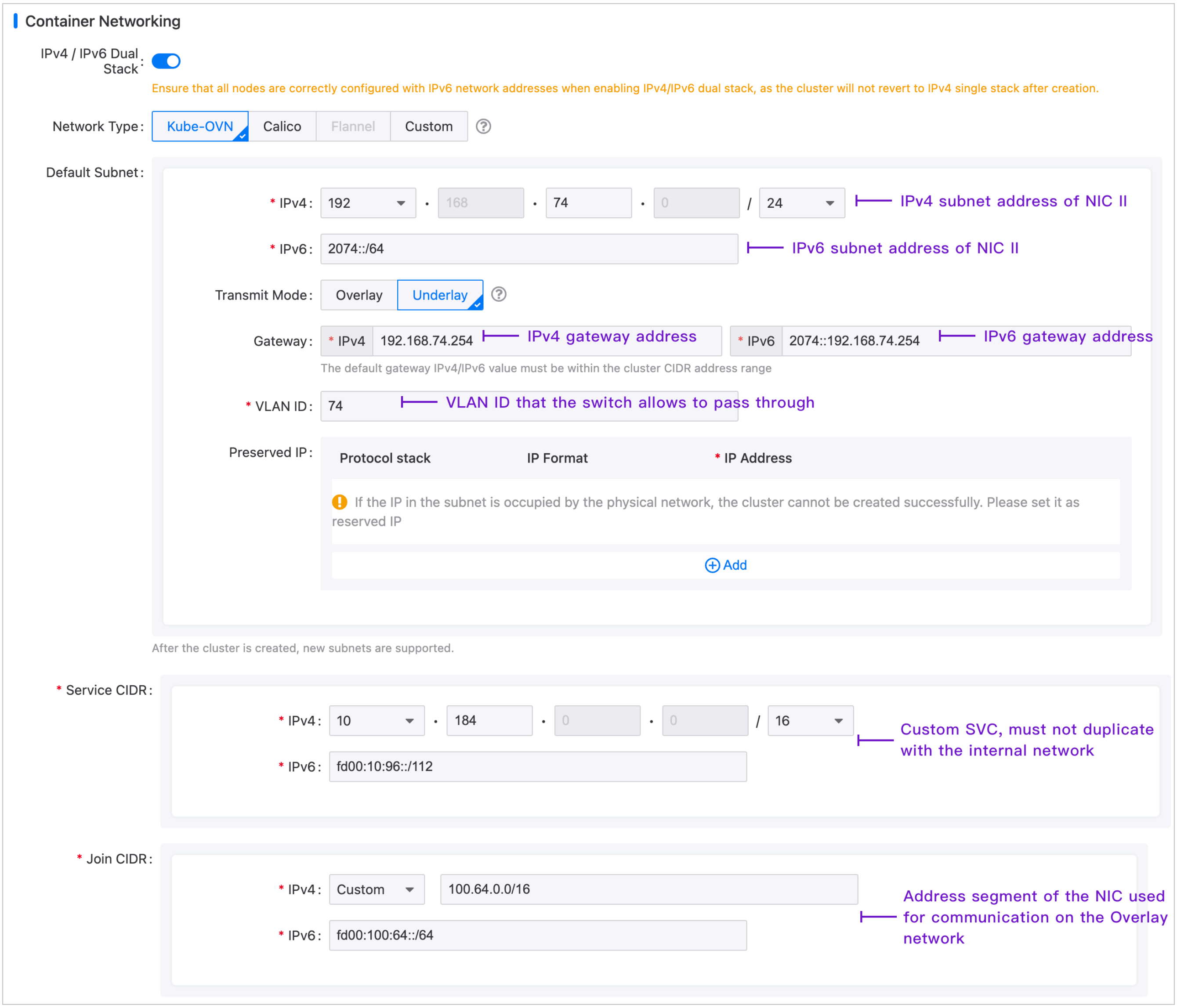
Test if NIC Two can communicate with the gateway address:

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

Platform Configuration

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

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



Container Networking

IPv4 / IPv6 Dual Stack: On

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

Network Type: [✓ Kube-OVN] Calico Flannel Custom

Default Subnet:

* IPv4: 192 . 168 . 74 . 0 / 24 — IPv4 subnet address of NIC II

* IPv6: 2074::/64 — IPv6 subnet address of NIC II

Transmit Mode: Overlay [✓ Underlay]

Gateway: * IPv4 192.168.74.254 — IPv4 gateway address * IPv6 2074::192.168.74.254 — IPv6 gateway address

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

* VLAN ID: 74 — VLAN ID that the switch allows to pass through

Preserved IP:

Protocol stack	IP Format	* IP Address

If the IP in the subnet is occupied by the physical network, the cluster cannot be created successfully. Please set it as reserved IP

Add

After the cluster is created, new subnets are supported.

* Service CIDR:

* IPv4: 10 . 184 . 0 . 0 / 16

* IPv6: fd00:10:96::/112

Custom SVC, must not duplicate with the internal network

* Join CIDR:

* IPv4: Custom 100.64.0.0/16

* IPv6: fd00:100:64::/64

Address segment of the NIC used for communication on the Overlay network

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

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`
   - `Docker (20.10.7)`
3. Ensure that the Docker service starts on boot for each host using the following command: `sudo systemctl enable docker.service`.
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 check script
      check_timeout: 3  # optional, default 3: check script timeout period
      name: "vip-1" # Identifier for this instance, can only contain alphanumeric
characters and hyphens, cannot start with a hyphen
      peer: [ "192.168.128.116", "192.168.128.75" ] # Keepalived node IP, actual
generated keepalived.conf will remove all IPs on the interface
https://github.com/osixia/docker-keepalived/issues/33
      kube_lock:
        kubecfgs: # The kube-config list used by kube-lock will sequentially attempt
these kubecfgs for leader election in Keepalived
          - "/live/cfg/kubecfg/kubecfg01.conf"
          - "/live/cfg/kubecfg/kubecfg02.conf"
          - "/live/cfg/kubecfg/kubecfg03.conf"
    ipvs: # Configuration for option IPVS
      ips: [ "192.168.143.192", "192.168.138.100","192.168.129.100" ] # IPVS backend,
change k8s master node IP to ALB node's node IP
      ports: # Configure health check logic for each port on the VIP
        - port: 80 # The port on the virtual server must match the real server's port
          virtual_server_config: |
            delay_loop 10  # Interval for performing health checks on the real server
            lb_algo rr
            lb_kind NAT
            protocol TCP
          raw_check: |
            TCP_CHECK {
                connect_timeout 10
                connect_port 1936
            }
  - vip:
      vip: 2004::192:168:128:118
      id: 102
      interface: "eth0"
      peer: [ "2004::192:168:128:75","2004::192:168:128:116" ]
      kube_lock:
        kubecfgs: # The kube-config list used by kube-lock will sequentially attempt
these kubecfgs for leader election in Keepalived
          - "/live/cfg/kubecfg/kubecfg01.conf"
          - "/live/cfg/kubecfg/kubecfg02.conf"
          - "/live/cfg/kubecfg/kubecfg03.conf"
    ipvs:
      ips: [ "2004::192:168:143:192","2004::192:168:138:100","2004::192:168:129:100" ]
      ports:
        - port: 80
          virtual_server_config: |
            delay_loop 10
            lb_algo rr
            lb_kind NAT
            protocol TCP
          raw_check: |
            TCP_CHECK {
                connect_timeout 1
                connect_port 1936
            }
```

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

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

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

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

8. Check the validity of the certificates.

   1. Copy `/usr/bin/kubectl` from the Master node of the business 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 Docker.

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

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

Verification

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

```
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight       ActiveConn InActConn
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="http://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>
```

```
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="http://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>
```

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

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.

Install Ingress-Nginx via Cluster Plugin

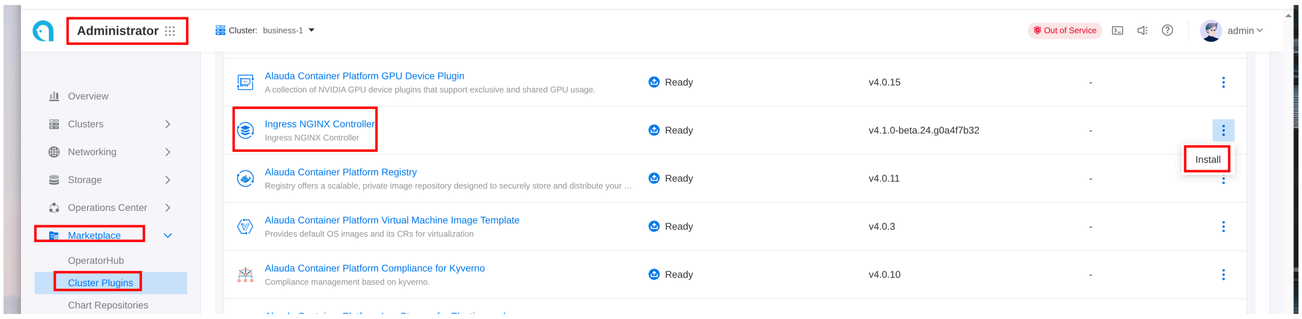
TOC

Overview

The NGINX Ingress Controller is deployed as a cluster plugin in the `cpaas-system` namespace. This guide covers installation, configuration, and best practices for managing the Ingress NGINX Controller in your Kubernetes cluster.

Installation

1. Navigate to `Administrator -> Marketplace -> Cluster Plugin`
2. Locate the Ingress NGINX plugin and click `Install`



Alauda Container Platform GPU Device Plugin A collection of NVIDIA GPU device plugins that support exclusive and shared GPU usage.	Ready	v4.0.15	-	⋮
Ingress NGINX Controller Ingress NGINX Controller	Ready	v4.1.0-beta.24.g0a4f7b32	-	⋮
Alauda Container Platform Registry Registry offers a scalable, private image repository designed to securely store and distribute your ...	Ready	v4.0.11	-	⋮
Alauda Container Platform Virtual Machine Image Template Provides default OS images and its CRs for virtualization	Ready	v4.0.3	-	⋮
Alauda Container Platform Compliance for Kyverno Compliance management based on kyverno.	Ready	v4.0.10	-	⋮

Configuration Management

Updating Configuration

1. Configure kubectl to use the **global** cluster context
2. Retrieve the ModuleInfo name for your cluster:

```
kubectl get minfo | grep ingress | grep $CLUSTER_NAME
```

3. Edit the ModuleInfo configuration:

```
kubectl edit minfo $MINFO
```

The `spec.config` section corresponds to the Ingress NGINX Helm chart values.

Common Configuration Scenarios

Exposing via LoadBalancer

To expose the Ingress Controller using a LoadBalancer service:

```yaml
spec:
  config:
    controller:
      service:
        type: LoadBalancer
```

MetalLB Integration

To specify LoadBalancer VIP when using MetalLB:

```yaml
spec:
  config:
    controller:
      service:
        annotations:
          metallb.universe.tf/loadBalancerIPs: "192.168.2.2"  # Desired VIP
          metallb.universe.tf/address-pool: "pool-name"        # MetalLB address pool
```

Advanced Controller Deployment Settings

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

```yaml
spec:
  config:
    controller:
      hostNetwork: false
      replicaCount: 1
      nodeSelector:
        kubernetes.io/os: linux
      resources:
        limits:
          cpu: 200m
          memory: 256Mi
        requests:
          cpu: 200m
          memory: 256Mi
```

SSL Passthrough

Enable SSL passthrough functionality:

```yaml
spec:
  config:
    controller:
      extraArgs:
        enable-ssl-passthrough: ""
```

IPv6 Single-Stack Support

Configure for IPv6-only environments:

```yaml
spec:
  config:
    controller:
      service:
        ipFamilies:
          - IPv6
```

Performance Tuning

Resource Allocation Guidelines

Small Scale (< 300 QPS)

```yaml
spec:
  config:
    controller:
      resources:
        limits:
          cpu: 200m
          memory: 256Mi
        requests:
          cpu: 200m
          memory: 256Mi
```

Medium Scale (< 10,000 QPS)

```yaml
spec:
  config:
    controller:
      resources:
        limits:
          cpu: "2"
          memory: 1Gi
        requests:
          cpu: "2"
          memory: 1Gi
```

Large Scale (< 20,000 QPS)

```yaml
spec:
  config:
    controller:
      resources:
        limits:
          cpu: "4"
          memory: 2Gi
        requests:
          cpu: "4"
          memory: 2Gi
```

High Performance (Unlimited)

For maximum performance with no CPU limits:

```yaml
spec:
  config:
    controller:
      resources:
        requests:
          cpu: "4"
          memory: 2Gi
```

Important Notes

Limitations

- Only one Ingress NGINX Controller instance is supported per cluster

Version Compatibility

Current version mapping: ACP Ingress-NGINX 4.1.x corresponds to the official chart 4.12.2 (controller v1.12.3)

Related Resources

tasks for ingress-nginx

official Ingress NGINX chart ↗

official Ingress NGINX documentation ↗

Tasks for Ingress-Nginx

TOC

Prerequisites

install ingress-nginx

Max Connections

max-worker-connections ↗

Timeout

config timeout ↗

Sticky Session

config sticky session ↗

Header Modification

Action	Link
set header in request	proxy-set-header ↗
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

config rate limiting ↗

WAF

modsecurity ↗

Forward-header control

x-forwarded-prefix-header ↗

HTTPS

TLS re-encrypt and verify backend certificate

verify backend https certificate ↗

TLS edge termination

backend protocol ↗

Passthrough

ssl-passthrough ↗

Default Certificate

default-ssl-certificate ↗

Menu

ALB

Auth

TOC

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-auth -o
jsonpath='{.items[*].status.podIP}');echo $ALB_IP
```

Configure Secret and Ingress

```
# echo "Zm9vOiRhcHIxJHFJQ05aNjFRJDJpb29pSlZVQU1tcHJxMjU4L0NoUDE=" | base64 -d #
foo:$apr1$qICNZ61Q$2iooiJVUAMmprq258/ChP1
# openssl passwd -apr1 -salt qICNZ61Q bar # $apr1$qICNZ61Q$2iooiJVUAMmprq258/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://$ALB_IP:80/app-file # should
return 200
# wrong password
curl -v -X GET -H "Authorization: Basic XXXXOmJhcg==" http://$ALB_IP:80/app-file # should
return 401
```

Related Ingress Annotations

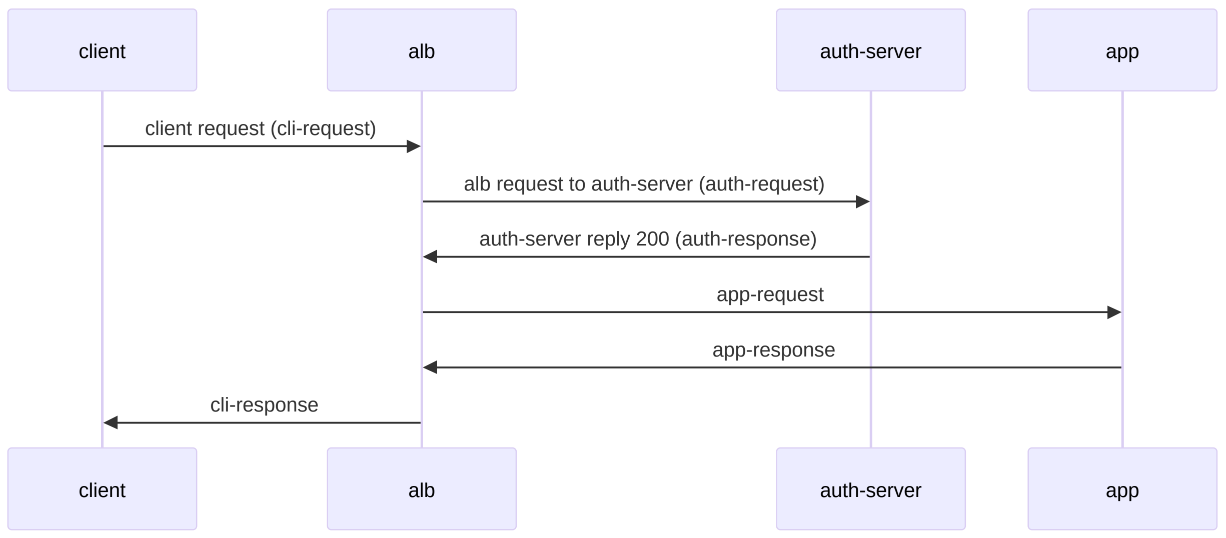
Ingress-nginx defines a series of annotations to configure the specific details of the authentication process. Below is a list of annotations that ALB supports, where "v" indicates support and "x" indicates no support.

	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	
nginx.ingress.kubernetes.io/auth-request-redirect	v	string	

	support	type	note
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
nginx.ingress.kubernetes.io/auth-keepalive	x	number	

	support	type	note
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


client
alb
auth-server
app
client request (cli-request)
alb request to auth-server (auth-request)
auth-server reply 200 (auth-response)
app-request
app-response
cli-response
client
alb
auth-server
app

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

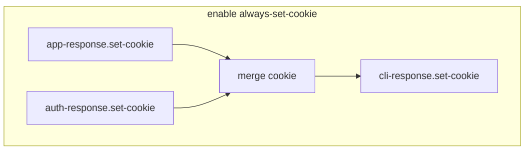
Value is a comma-separated string, allowing us to bring specific headers from auth-response to app-request. example:

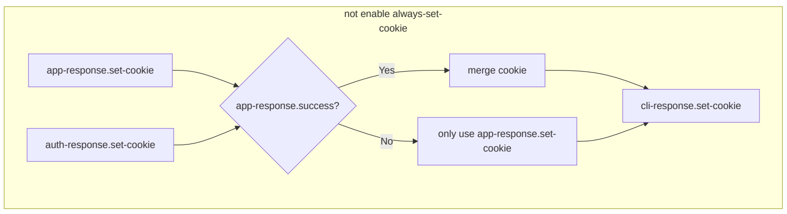
```
nginx.ingress.kubernetes.io/auth-response-headers: Remote-User,Remote-Name
```

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

cookie handling

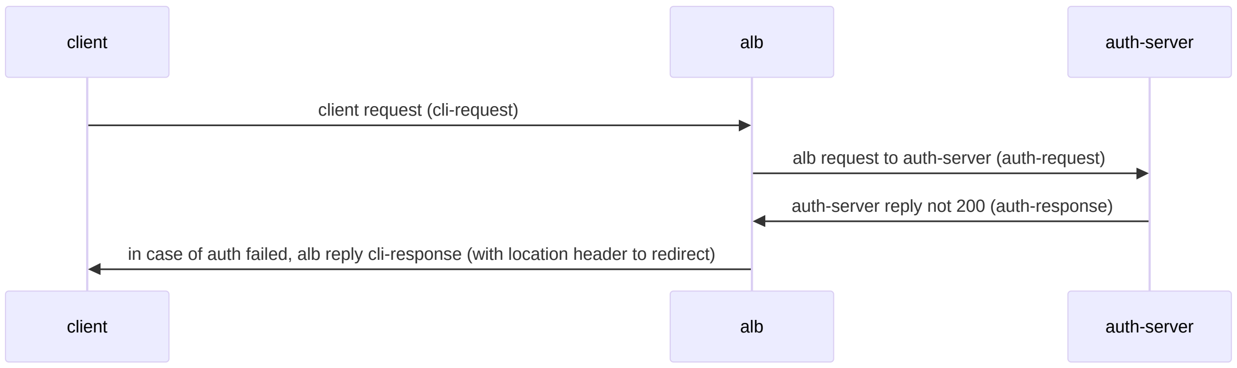
auth-response and app-response can both set cookies. By default, only when app-response.success, the auth-response.set-cookie will be merged into cli-response.set-cookie.





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.


client
alb
auth-server
client request (cli-request)
alb request to auth-server (auth-request)
auth-server reply not 200 (auth-response)
in case of auth failed, alb reply cli-response (with location header to redirect)
client
alb
auth-server

auth-signin

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

auth-signin-redirect-param

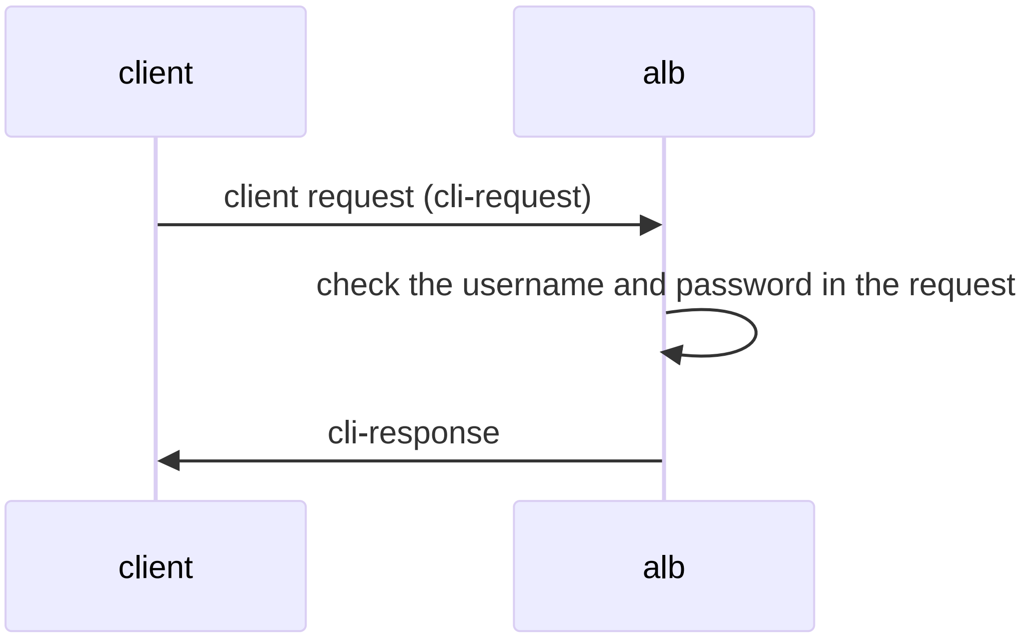
The name of the query parameter in the signin-url, default is rd. if the signin-url does not contain the `auth-signin-redirect-param` specified parameter name, alb will automatically add the parameter. The parameter value will be set to `$pass_access_scheme://$http_host$escaped_request_uri`, used to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


client
alb
client request (cli-request)
check the username and password in the request
cli-response
client
alb

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.

```yaml
auth:
  # Basic authentication configuration
  basic:
    #  string; corresponding to nginx.ingress.kubernetes.io/auth-type: basic
    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-type
    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-set-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-redirect-param
    signin_redirect_param: "redirect_to"
    #  []string; corresponding to nginx.ingress.kubernetes.io/auth-response-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

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

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 business 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.
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.

Header Modification

TOC

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 variable 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":"cookie_auth_token"}}'
```

To set HSTS, you can use:

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

HTTP Redirect

TOC

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

Annotation	Description
nginx.ingress.kubernetes.io/temporal-redirect-code	Corresponds to code in CR
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.

Menu

L4/L7 Timeout

TOC

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.

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

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	Explanation
nginx.ingress.kubernetes.io/enable-modsecurity	bool	Ingress	Enable ModSecurity.
nginx.ingress.kubernetes.io/enable-owasp-core-rules	bool	Ingress	Enable the OWASP Core Rule Set.
nginx.ingress.kubernetes.io/modsecurity-transaction-id	string	Ingress	Used to identify unique transaction IDs for each request, aiding in logging and debugging.
nginx.ingress.kubernetes.io/modsecurity-snippet	string	Ingress, ALB, FT, Rule	Allows users to insert custom ModSecurity configurations

Annotation	Type	Applicable Object	Explanation
			to meet specific security requirements

- **ALB Special Annotations**

Annotation	Type	Applicable Object	Explanation
alb.modsecurity.cpaas.io/use-recommend	bool	Ingress	Enable or disable recommended ModSecurity rules; set to `true` to apply a predefined set of security rules.
alb.modsecurity.cpaas.io/cmref	string	Ingress	Reference specific configurations, e.g., custom security configurations can be loaded by specifying the ConfigMap's reference path (`$ns/$name#$section`).

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
```

```yaml
  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: docker.io/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
```

TCP/HTTP Keepalive

TOC

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

```yaml
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_KEEPALIVE)
      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`.

Use OAuth Proxy with ALB

TOC

Overview

This document demonstrates how to use OAuth Proxy with ALB to implement external authentication.

Procedure

Follow these steps to use the feature:

1. Deploy kind

```
kind create cluster --name alb-auth --image=kindest/node:v1.28.0
kind get kubeconfig --name=alb-auth > ~/.kube/config
```

2. Deploy alb

```
helm repo add alb https://alauda.github.io/alb/;helm repo update;helm search repo|grep
alb
helm install alb-operator alb/alauda-alb2
alb_ip=$(docker inspect -f '{{range.NetworkSettings.Networks}}{{.IPAddress}}{{end}}'
alb-auth-control-plane)
echo $alb_ip
cat <<EOF | kubectl apply -f -
apiVersion: crd.alauda.io/v2
kind: ALB2
metadata:
    name: alb-auth
spec:
    address: "$alb_ip"
    type: "nginx"
    config:
        networkMode: host
        loadbalancerName: alb-demo
        projects:
        - ALL_ALL
        replicas: 1
EOF
```

3. Deploy test application

- Create github oauth app ↗

  > Note that `$GITHUB_CLIENT_ID` `$GITHUB_CLIENT_SECRET` will be obtained in this step, which needs to be set in the environment variable

- Configure dns

  > Here we use echo.com as the application domain, auth.alb.echo.com and alb.echo.com

- Deploy oauth-proxy

  > oauth2-proxy needs to access github, which may require setting the HTTPS_PROXY environment variable

```
COOKIE_SECRET=$(python -c 'import os,base64;
print(base64.urlsafe_b64encode(os.urandom(32)).decode())')
OAUTH2_PROXY_IMAGE="quay.io/oauth2-proxy/oauth2-proxy:v7.7.1"
kind load docker-image $OAUTH2_PROXY_IMAGE --name alb-auth
cat <<EOF | kubectl apply -f -
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    k8s-app: oauth2-proxy
  name: oauth2-proxy
spec:
  replicas: 1
  selector:
    matchLabels:
      k8s-app: oauth2-proxy
  template:
    metadata:
      labels:
        k8s-app: oauth2-proxy
    spec:
      containers:
        - args:
            - --http-address=0.0.0.0:4180
            - --redirect-url=http://auth.alb.echo.com/oauth2/callback
            - --provider=github
            - --whitelist-domain=.alb.echo.com
            - --email-domain=*
            - --upstream=file:///dev/null
            - --cookie-domain=.alb.echo.com
            - --cookie-secure=false
            - --reverse-proxy=true
          env:
            - name: OAUTH2_PROXY_CLIENT_ID
              value: $GITHUB_CLIENT_ID
            - name: OAUTH2_PROXY_CLIENT_SECRET
              value: $GITHUB_CLIENT_SECRET
            - name: OAUTH2_PROXY_COOKIE_SECRET
              value: $COOKIE_SECRET
          image: $OAUTH2_PROXY_IMAGE
          imagePullPolicy: IfNotPresent
          name: oauth2-proxy
          ports:
```

```
          - containerPort: 4180
            name: http
            protocol: TCP
          - containerPort: 44180
            name: metrics
            protocol: TCP
---
apiVersion: v1
kind: Service
metadata:
  labels:
    k8s-app: oauth2-proxy
  name: oauth2-proxy
spec:
 ports:
 - appProtocol: http
   name: http
   port: 80
   protocol: TCP
   targetPort: http
 - appProtocol: http
   name: metrics
   port: 44180
   protocol: TCP
   targetPort: metrics
 selector:
   k8s-app: oauth2-proxy
EOF
```

4. Configure ingress

   We will configure two ingresses, auth.alb.echo.com and alb.echo.com

```
cat <<EOF | kubectl apply -f -
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    nginx.ingress.kubernetes.io/auth-url: "https://auth.alb.echo.com/oauth2/auth"
    nginx.ingress.kubernetes.io/auth-signin: "https://auth.alb.echo.com/oauth2/start?
rd=http://\$host\$request_uri"
  name: echo-resty
spec:
  ingressClassName: alb-auth
  rules:
    - host: alb.echo.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: echo-resty
                port:
                  number: 80
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: oauth2-proxy
spec:
  ingressClassName: alb-auth
  rules:
    - host: auth.alb.echo.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: oauth2-proxy
                port:
                  number: 80
EOF
```

Result

- After the operation is complete, an alb, oauth-proxy, and test application will be deployed.
- After accessing alb.echo.com, you will be redirected to the github authentication page, and after verification, you can see the output of the application

Configure GatewayApi Gateway via ALB

An inbound gateway (Gateway) is an instance deployed from the Gateway Class. It creates listeners to capture external traffic on specified domain names and ports. Together with routing rules, it can route the specified external traffic to the corresponding backend instances.

Create an inbound gateway to enable more granular allocation of network resources.

TOC

Terminology

Resource Name	Overview	Usage Instructions
Gateway Class	In the standard Gateway API documentation, the Gateway Class is defined as a template for creating gateways. Different templates can create inbound gateways for different business scenarios, facilitating rapid traffic management.	The platform includes dedicated Gateway Classes.
Inbound Gateway	The inbound gateway corresponds to specific resource instances, and users can exclusively utilize all listening and computing resources of this inbound gateway. It is a configuration of routing rules effective for the listener. When external traffic is detected by the gateway, it will be distributed to backend instances according to the routing rules.	It can be viewed as a load balancer instance.
Route Rule	Route rules define a series of guidelines for traffic distribution from the gateway to services. The currently standard supported types of routing rules in the Gateway API include HTTPRoute, TCPRoute, UDPRoute, etc.	The platform currently supports listening to HTTP, HTTPS, TCP, and UDP protocols.

Prerequisites

The platform administrator must ensure that the cluster supports LoadBalancer type internal routing. For public cloud clusters, the LoadBalancer Service Controller must be installed. In non-public cloud clusters, the platform provides the external address pool feature, which allows LoadBalancer type internal routing to automatically obtain an IP from the external address pool for external access after configuration is complete.

Example Gateway and Alb2 custom resource (CR)

```
# demo-gateway.yaml
apiVersion: gateway.networking.k8s.io/v1beta1
kind: Gateway
metadata:
  namespace: k-1
  name: test
  annotations:
    cpaas.io/display-name: ces
  labels:
    alb.cpaas.io/alb-ref: test-o93q7 1
spec:
  gatewayClassName: exclusive-gateway 2
  listeners:
    - allowedRoutes:
        namespaces:
          from: All
      name: gateway-metric
      protocol: TCP
      port: 11782
---
apiVersion: crd.alauda.io/v2beta1
kind: ALB2
metadata:
  namespace: k-1
  name: test-o93q7 3
spec:
  type: nginx
  config:
    enableAlb: false
    networkMode: container
    resources:
      limits:
        cpu: 200m
        memory: 256Mi
      requests:
        cpu: 200m
        memory: 256Mi
    vip:
      enableLbSvc: true
      lbSvcAnnotations: {}
    gateway:
      mode: standalone
      name: test 4
```

1. Which ALB this gateway uses.
2. See the Gateway Class introduction below.
3. `ALB2` name format: `{gatewayName}-{random}`.
4. Gateway name.

Creating Gateway by using the web console

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

Parameter	Description
Name	The name of the inbound gateway.
Gateway Class	The gateway class defines the behavior of the gateway, similar to the concept of storage classes (StorageClasses); it is a cluster resource. **Dedicated**: The inbound gateway will correspond to a specific resource instance, and the user can utilize all listeners and computing resources of this gateway.
Specification	You can choose the recommended usage scenario based on your needs or customize the resource limits.
Access Address	The address of the inbound gateway, which is automatically obtained by default.
Internal Routing Annotation	Used to declare the configuration or capabilities for LoadBalancer type internal routing. For specific annotation information, please refer to LoadBalancer type internal routing annotation instructions.

5. Click **Create**.

Creating Gateway by using the CLI

```
kubectl apply -f demo-gateway.yaml
```

Viewing Resources Created by the Platform

After the inbound gateway is created, the platform automatically creates many resources. Do not delete the resources below.

Default Created Resources	Name
ALB2 Type Resource	*name-lb-random*
Deployment	*name-lb-random*
Internal Routing	• *name-lb-random* • *name-lb-random-lb-random*
Configuration Dictionary	• *name-lb-random-port-info* • *name-lb-random*
Service Account	*name-lb-random-serviceaccount*

Updating Gateways

NOTE

Updating the inbound gateway will cause a service interruption of 3-5 minutes. Please choose an appropriate time for this operation.

Updating Gateway by using the web console

1. Access the **Container Platform**.
2. In the left navigation bar, click **Network** > **Inbound Gateway**.
3. Click ⋮ > **Update**.
4. Update the inbound gateway configuration as needed.

   Note: Please set the specifications reasonably based on business requirements.
5. Click **Update**.

Add Listener

Monitor traffic under specified domain names and forward it to backend instances according to the bound routing rules.

Prerequisites

- If you need to monitor HTTP protocol, please contact the administrator in advance to prepare the **domain name**.
- If you need to monitor HTTPS protocol, please contact the administrator in advance to prepare the **domain name** and **certificate**.

Add Listener by using the web console

1. In the left navigation bar, click **Network** > **Inbound Gateway**.

2. Click ***Inbound Gateway Name***.

3. Click **Add Listener**.

4. Refer to the following instructions to configure specific parameters.

Parameter	Description
Listener Protocol and Port	Currently supports monitoring HTTP, HTTPS, TCP, and UDP protocols, and you can custom input the port to be monitored, for example: `80`. **Note**: • When the ports are the same, HTTP, HTTPS, and TCP listener types cannot coexist; you can only select one of the protocols. • When using HTTP or HTTPS protocol, if the ports are the same, the domain names must be different.
Domain Name	Select an available domain name in the current namespace, used to monitor network traffic accessing this domain name. **Hint**: TCP and UDP protocols do not support selecting domain names.

5. Click **Create**.

Add Listener by using the CLI

```
kubectl patch gateway test \
  -n k-1 \
  --type=merge \
  -p '{
    "spec": {
      "listeners": [
        {
          "allowedRoutes": {
            "namespaces": {
              "from": "All"
            }
          },
          "name": "gateway-metric",
          "protocol": "TCP",
          "port": 11782
        },
        {
          "allowedRoutes": {
            "namespaces": {
              "from": "All"
            }
          },
          "name": "demo-listener",
          "protocol": "HTTP",
          "port": 8088,
          "hostname": "developer.test.cn"
        }
      ]
    }
  }'
```

Creating Route Rules

Route rules provide routing policies for incoming traffic, similar to inbound rules (Kubernetes Ingress). They expose network traffic monitored by the gateway to the internal routing of the cluster (Kubernetes Service), facilitating routing forwarding strategies. The key difference is that they target different service objects: inbound rules serve the Ingress Controller, while route rules serve the Ingress Gateway.

Once the listening is set up in the ingress gateway, the gateway will monitor traffic from specified domains and ports in real-time. The route rules can forward the incoming traffic to backend instances as desired.

Example HTTPRoute custom resource (CR)

```yaml
# example-httproute.yaml
apiVersion: gateway.networking.k8s.io/v1beta1
kind: HTTPRoute 1
metadata:
  namespace: k-1
  name: example-http-route
  annotations:
    cpaas.io/display-name: ""
spec:
  hostnames:
    - developer.test.cn
  parentRefs:
    - kind: Gateway
      namespace: k-1
      name: test
      sectionName: demo-listener 2
  rules:
    - matches:
        - path:
            type: Exact
            value: "/demo"
      filters: []
      backendRefs:
        - kind: Service
          name: test-service
          namespace: k-1
          port: 80
          weight: 100
```

1. The available types are: `HTTPRoute` , `TCPRoute` , `UDPRoute` .
2. `Gateway` listener name.

NOTE

If there is no matching rule for the **Path** object in the HTTPRoute type route rule, a matching rule with PathPrefix mode and a value of / will be automatically added.

Creating Route by using the web console

1. Access the **Container Platform**.
2. In the left navigation bar, click **Network** > **Route Rules**.
3. Click **Create Route Rule**.
4. Follow the instructions below to configure some parameters.

Parameter	Description
Route Type	The currently supported route types are: HTTPRoute, TCPRoute, UDPRoute. **Tip:** HTTPRoute supports publishing to HTTP and HTTPS protocol listeners.
Publish to Listener	In the left selection box, select the created **Ingress Gateway**, and in the right selection box, select the created **Listener**. The platform will publish the created route rules to the listener below, enabling the gateway to forward captured traffic to specified backend instances. **Note:** It is not allowed to publish route rules to a listener that is on port **11782** or has already mounted TCP or UDP routes.
Match	You can add one or more matching rules to capture traffic that meets the requirements. For example, **capture traffic with specified Path**, **capture traffic with specified method**, etc. **Note:**

Parameter	Description
	<ul><li>Click **Add**; when adding multiple route rules, the relationship between the rules is 'AND', and all rules must be matched to be effective.</li><li>Click **Add Match**; when adding multiple groups of route rules, the relationship between the groups is 'OR', and any group matching can be effective.</li><li>TCPRoute and UDPRoute do not support configuring match rules.</li><li>When the matching object is **path**, and the matching method is **Exact** or **PathPrefix**, the input **value** must start with "/" and disallow characters like "//", "/./", "/../", "%2f", "%2F", "#", "/..", "/." etc.</li></ul>
Action	You can add one or more actions to process the captured traffic.<ul><li>Header: The header of the HTTP message contains much metadata that provides additional information about the request or response. By modifying header fields, the server can influence how the request and response are processed.</li><li>Redirect: The matched URL will be processed in the specified manner, then the request will be initiated again.</li><li>Rewrite: The matched URL will be processed in the specified manner, then the request will be redirected to a different resource path or filename.</li></ul>**Note:**<ul><li>Click **Add**; when adding multiple action rules, the platform will execute all actions in order based on the displayed sequence of the rules.</li><li>TCPRoute and UDPRoute do not support configuring action rules.</li></ul>

Parameter	Description
	• Within the same route rule, there cannot be multiple **Header** type actions with the same **value**. • Within the same route rule, only one type of either **Redirect** or **Rewrite**, and only one mode of either **FullPath** or **PrefixPath** can exist. • If you wish to use the **PrefixPath** operation, please first add a matching rule of **PathPrefix** mode.
Backend Instance	After the rule takes effect, it will forward to the backend instance according to the selected internal routes and ports in the current namespace. You can also set weights, with higher weight values resulting in a higher probability of being polled. **Tip:** The percentage next to the weight indicates the probability of forwarding to that instance, calculated as the ratio of the current weight value to the sum of all weight values.

5. Click **Create**.

Creating Route by using the CLI

```
kubectl apply -f example-httproute.yaml
```

Bind NIC in ALB

By default, ALB listens on `0.0.0.0` for ipv4 and `::` for ipv6. In certain security scenarios, it needs to be bound to a specific network interface card (NIC).

TOC

For Cluster Embedded ALB

By default, an embedded ALB will be deployed in each cluster. In the global cluster, it should be named 'global-alb2', while in other clusters, it should be named 'cpaas-system'.

Replace `$CLUSTER` and `$NIC` with the actual cluster and NIC. If you use Alive (Alauda Container Platform Virtual IP Management), you need add `alive` to the nic list.

```
kubectl annotate cluster -n cpaas-system $cluster cpaas.io/alb-bind-nic='{"nic":
["$NIC","alive"]}'
```

By default, ALB enables IPv6 in a single-stack cluster. However, when using bindnic, the specified NIC might not have an IPv6 address. In such cases, ALB will still attempt to bind to `::*`. As a workaround, you could disable IPv6.

```
kubectl annotate cluster -n cpaas-system $cluster cpaas.io/alb-enable-ipv6='"false"'
```

For User-defined ALB

```
kubectl patch alb2 -n cpaas-system $ALB -p '{"spec":{"config":
{"enableIPV6":"false","bindNIC":"{\"nic\":[\"$NIC\",\"alive\"]}"}}}' --type=merge
```

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.

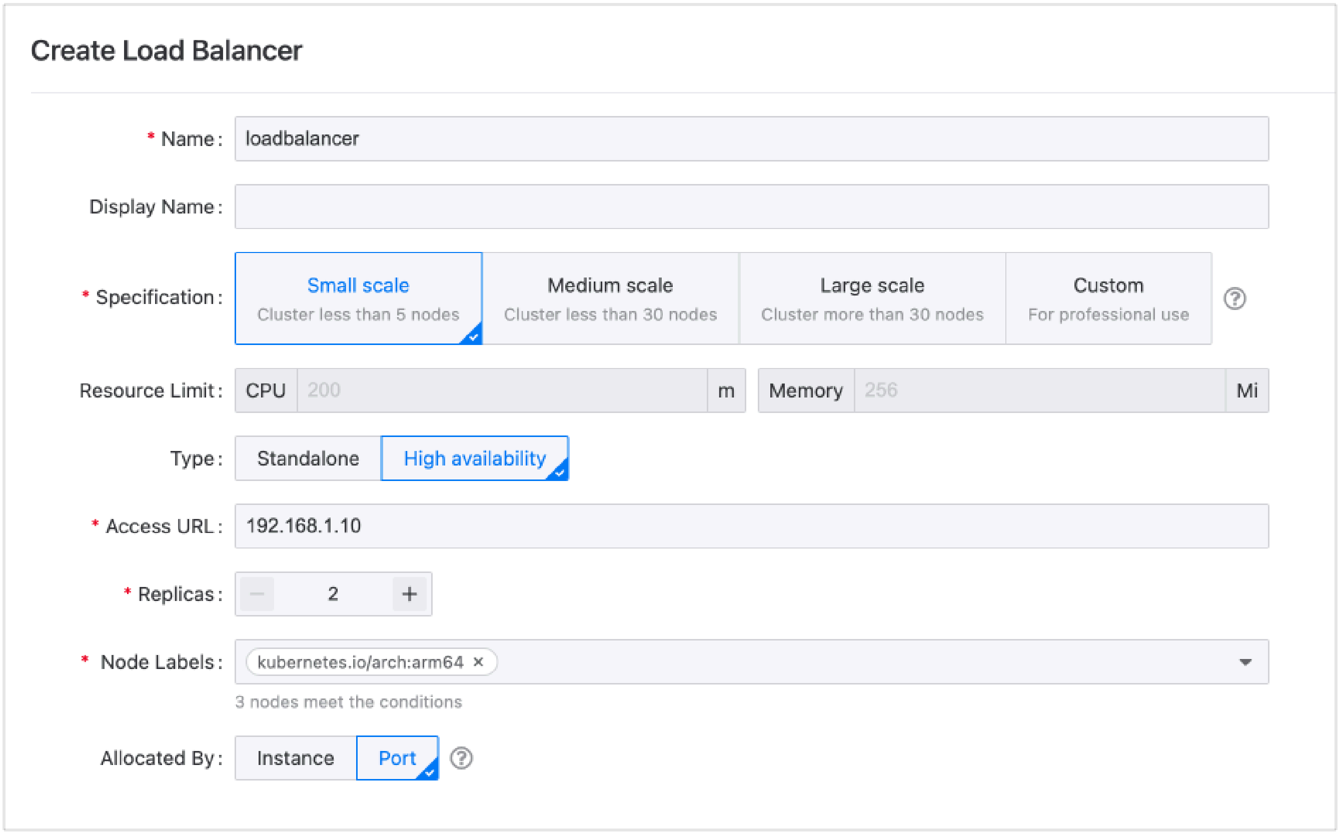
TOC

Small Production Environment

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.


Create Load Balancer

* Name: loadbalancer

Display Name:

* Specification: [✓ Small scale — Cluster less than 5 nodes] / Medium scale — Cluster less than 30 nodes / Large scale — Cluster more than 30 nodes / Custom — For professional use

Resource Limit: CPU 200 m / Memory 256 Mi

Type: Standalone / [✓ High availability]

* Access URL: 192.168.1.10

* Replicas: 2

* Node Labels: kubernetes.io/arch:arm64

3 nodes meet the conditions

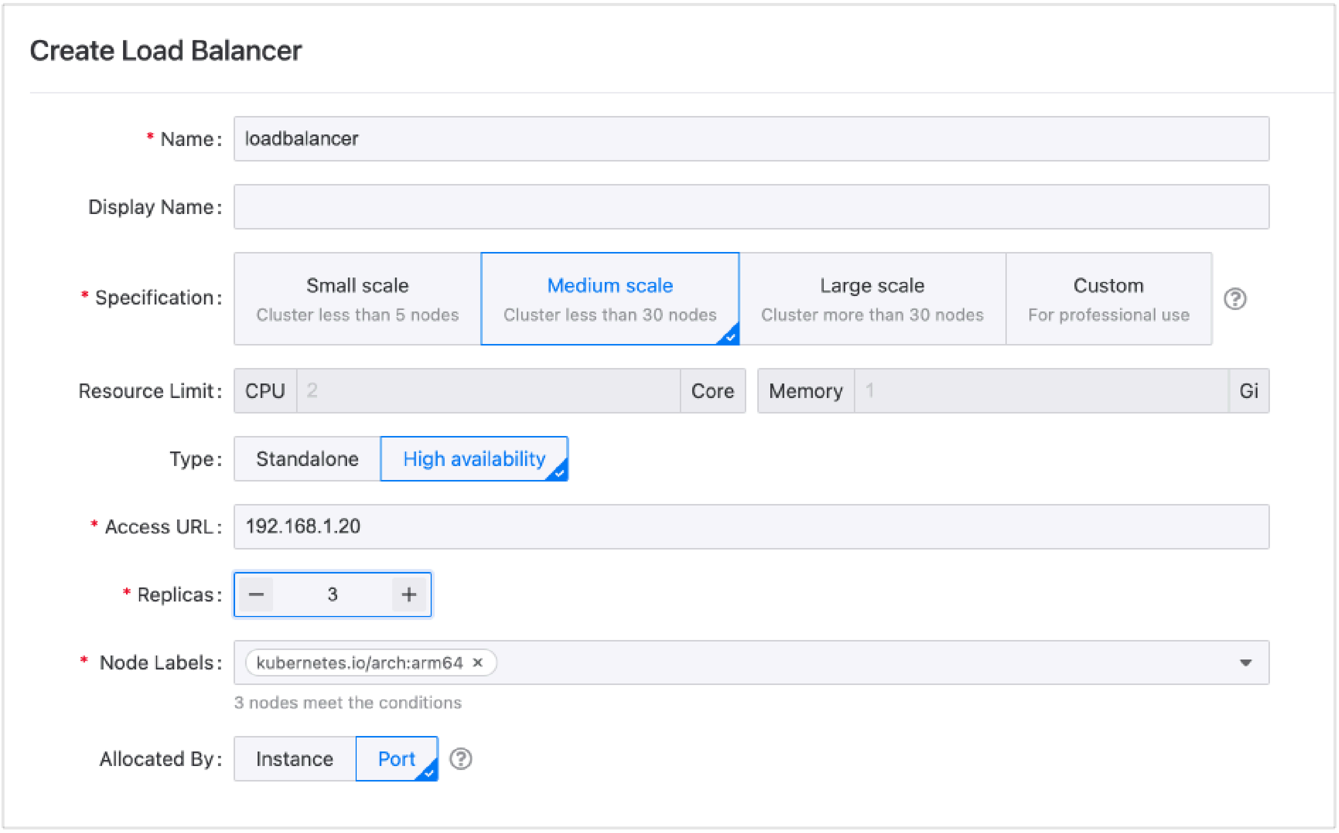
Allocated By: Instance / [✓ Port]

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.


Create Load Balancer

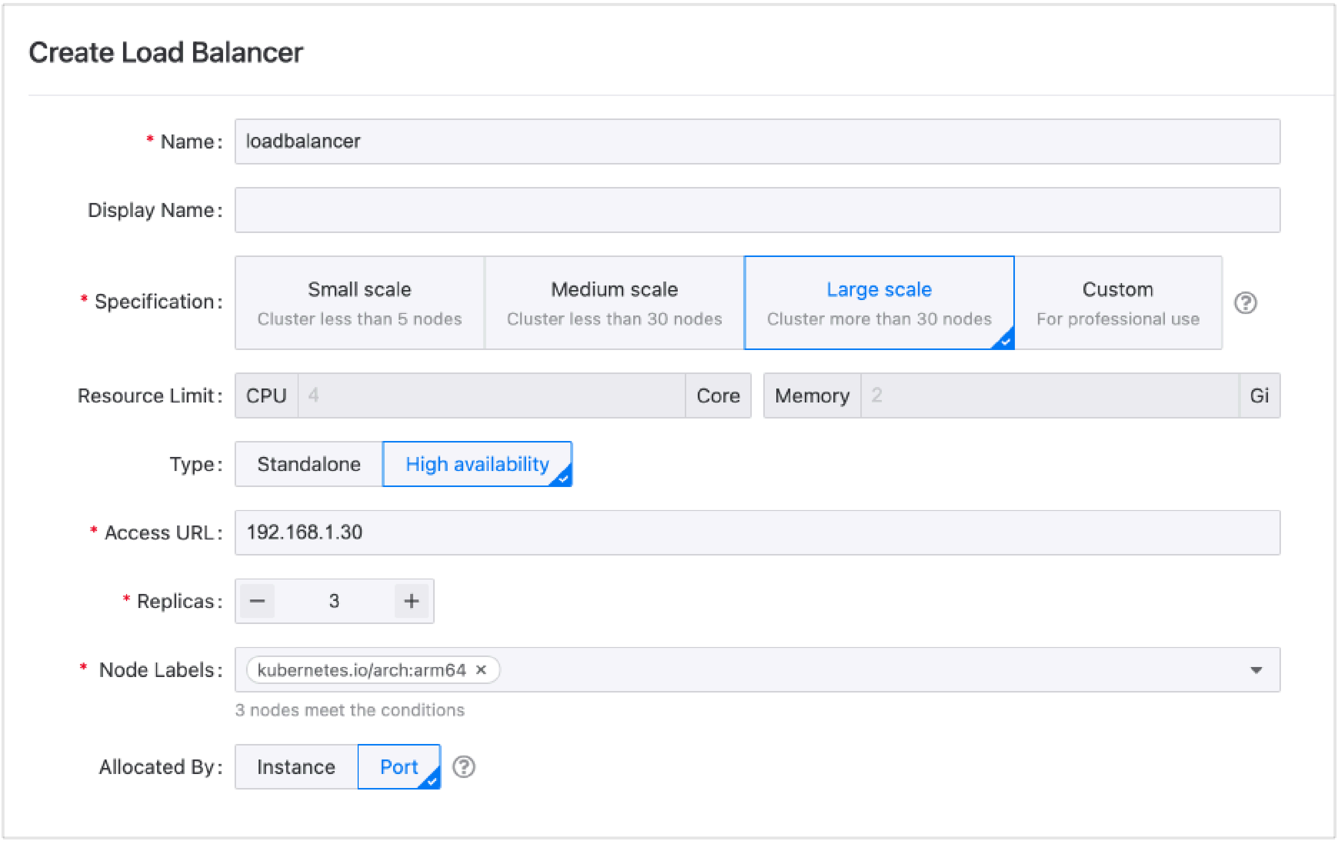
* Name:	loadbalancer
Display Name:	
* Specification:	Small scale (Cluster less than 5 nodes) / [✓ Medium scale (Cluster less than 30 nodes)] / Large scale (Cluster more than 30 nodes) / Custom (For professional use)
Resource Limit:	CPU 2 Core; Memory 1 Gi
Type:	Standalone [✓ High availability]
* Access URL:	192.168.1.20
* Replicas:	3
* Node Labels:	kubernetes.io/arch:arm64 × — 3 nodes meet the conditions
Allocated By:	Instance [✓ Port]

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.



Create Load Balancer

* Name:	loadbalancer
Display Name:	
* Specification:	Small scale (Cluster less than 5 nodes) / Medium scale (Cluster less than 30 nodes) / [✓ Large scale (Cluster more than 30 nodes)] / Custom (For professional use)
Resource Limit:	CPU 4 Core, Memory 2 Gi
Type:	Standalone / [✓ High availability]
* Access URL:	192.168.1.30
* Replicas:	3
* Node Labels:	kubernetes.io/arch:arm64 × 3 nodes meet the conditions
Allocated By:	Instance / [✓ Port]

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

Scenario	Deployment Recommendations
	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.

Usage Mode	Advantages	Disadvantages
		• 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.
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.

Deploy ALB

TOC

ALB

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

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

Prerequisites

The high availability of the **Load Balancer** requires a VIP. Please refer to Configure VIP.

Configure ALB

There are three parts to an ALB configuration.

```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 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

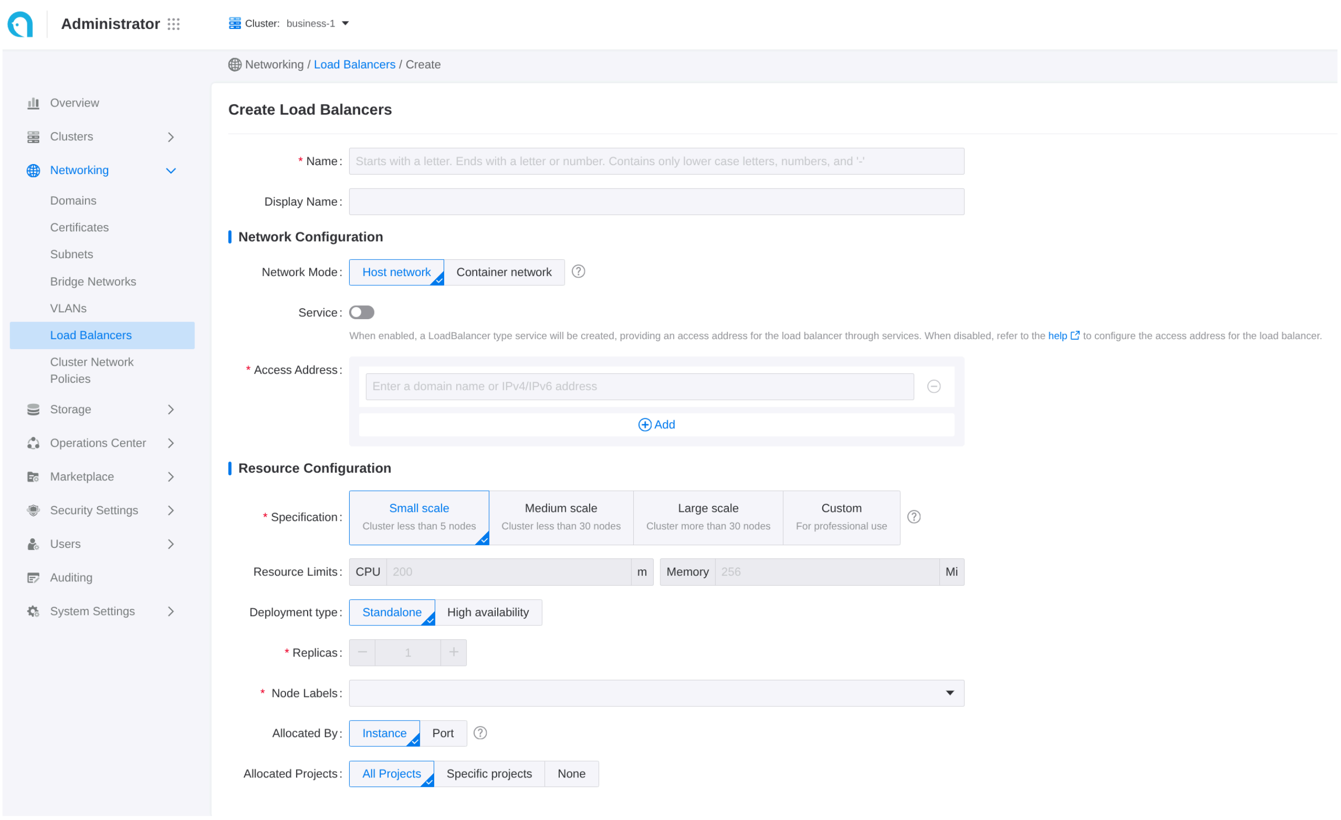
there are some global config which can be tweaked in alb cr.

- bind-nic
- ingress-sync

Operation On ALB

Creating

Using the web console.



Networking / Load Balancers / Create

Create Load Balancers

* Name: Starts with a letter. Ends with a letter or number. Contains only lower case letters, numbers, and '-'

Display Name:

Network Configuration

Network Mode: [✓ Host network] Container network

Service:

When enabled, a LoadBalancer type service will be created, providing an access address for the load balancer through services. When disabled, refer to the help to configure the access address for the load balancer.

* Access Address: Enter a domain name or IPv4/IPv6 address

Add

Resource Configuration

* Specification: [✓ Small scale Cluster less than 5 nodes] Medium scale Cluster less than 30 nodes Large scale Cluster more than 30 nodes Custom For professional use

Resource Limits: CPU 200 m Memory 256 Mi

Deployment type: [✓ Standalone] High availability

* Replicas: 1

* Node Labels:

Allocated By: [✓ Instance] Port

Allocated Projects: [✓ All Projects] Specific projects None

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
```

Listener Ports (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`:

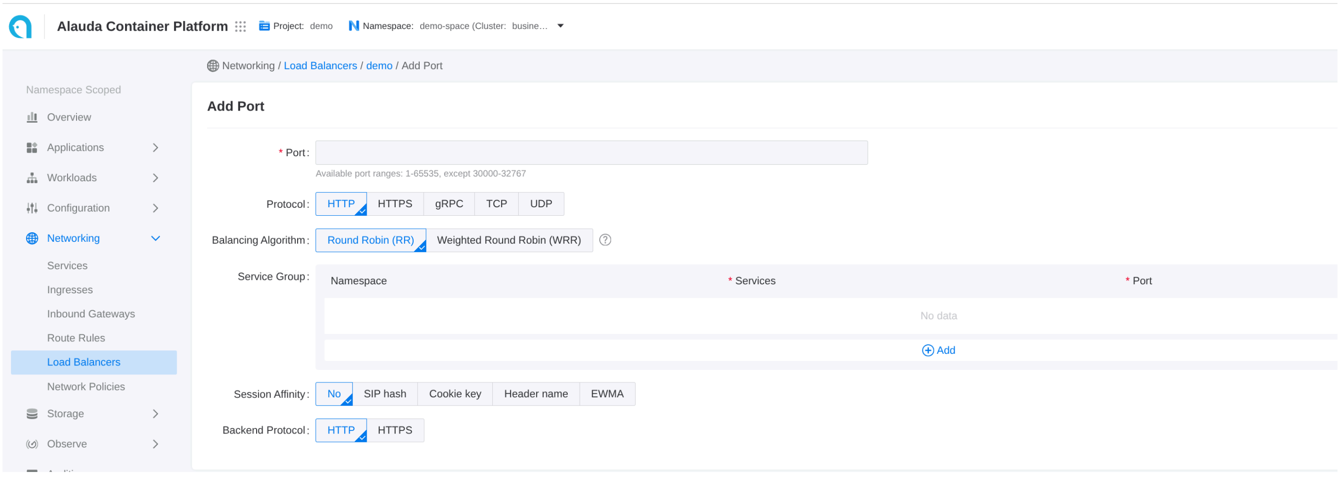
- L4 proxy: required. ALB forwards traffic to the default service group directly.
- L7 proxy: optional. ALB first matches Rules on this Frontend; if none match, it falls back to the default `serviceGroup`.

8 session_affinity_policy

Operation On Frontend

Creating

using the web console



Add Port

* Port:

Available port ranges: 1-65535, except 30000-32767

Protocol: [✓ HTTP] HTTPS gRPC TCP UDP

Balancing Algorithm: [✓ Round Robin (RR)] Weighted Round Robin (WRR)

Service Group:

Namespace	* Services	* Port
No data		

Add

Session Affinity: [✓ No] SIP hash Cookie key Header name EWMA

Backend Protocol: [✓ HTTP] HTTPS

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 webui 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.

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

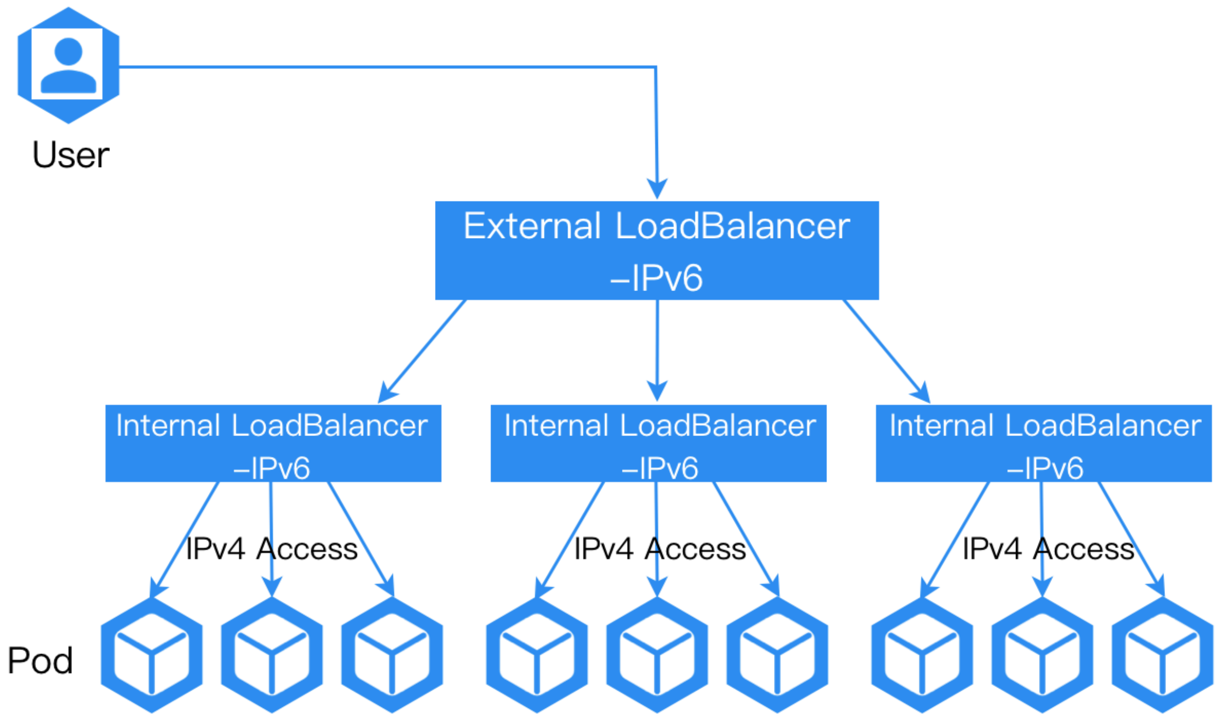
The cluster where the load balancer is located must deploy monitoring services.

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.

Forwarding IPv6 Traffic to IPv4 Addresses within the Cluster via ALB

By configuring an external load balancer for the cluster, we can forward IPv6 traffic to the internal IPv4 addresses within the cluster. This allows us to introduce IPv6 capabilities over the existing IPv4 network, providing greater flexibility and scalability to our system architecture, and better addressing diverse network demands.


User
External LoadBalancer –IPv6
Internal LoadBalancer –IPv6
Internal LoadBalancer –IPv6
Internal LoadBalancer –IPv6
IPv4 Access
IPv4 Access
IPv4 Access
Pod

TOC

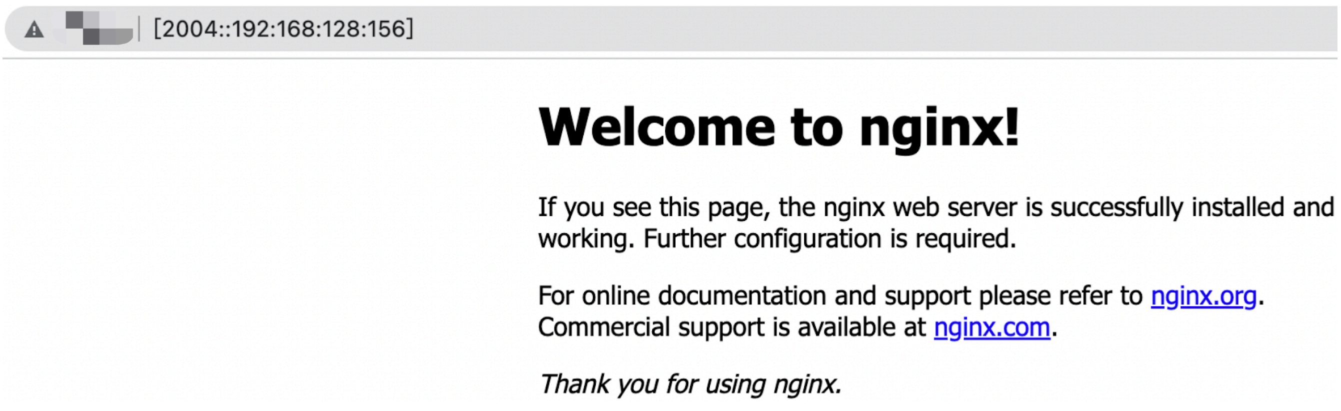
Configuration Method

1. Configure the IPv6 address for the node where the load balancer is located.
2. Ensure that the external load balancer has an IPv6 address, and make sure that traffic accessing the load balancer's IPv6 address can be forwarded to the IPv6 address of the node where the load balancer resides.

Once the above configuration is completed, the IPv4 services mounted on the load balancer can provide external IPv6 access capabilities through the load balancer.

Result Verification

After the configuration, accessing the IPv6 address of the external load balancer should allow normal access to the application.


[2004::192:168:128:156]

Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

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

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, where the name of the ALB is replaced with `<otel-alb>` in this document. For instructions on creating an ALB, refer to Deploy 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 the Master 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:

```yaml
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 the cluster Master node, 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.versions[2].schema.openAPIV3Schema.properties.spec.properties.config.properties.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.

Parameter	Description
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.
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

Parameter	Description
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
---
```

```yaml
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"
```

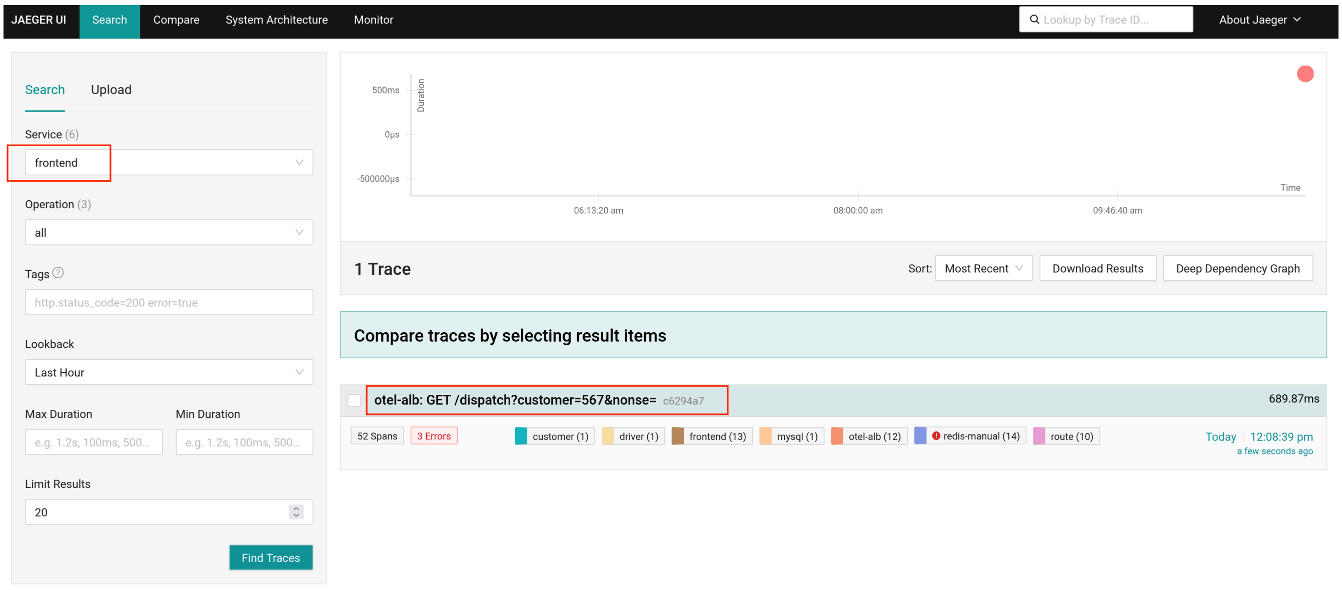
4. Execute the following command to obtain the access address of otel-alb.

```
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.



Service (6): frontend

Operation (3): all

Tags: http.status_code=200 error=true

Lookback: Last Hour

Max Duration: e.g. 1.2s, 100ms, 500...

Min Duration: e.g. 1.2s, 100ms, 500...

Limit Results: 20

1 Trace

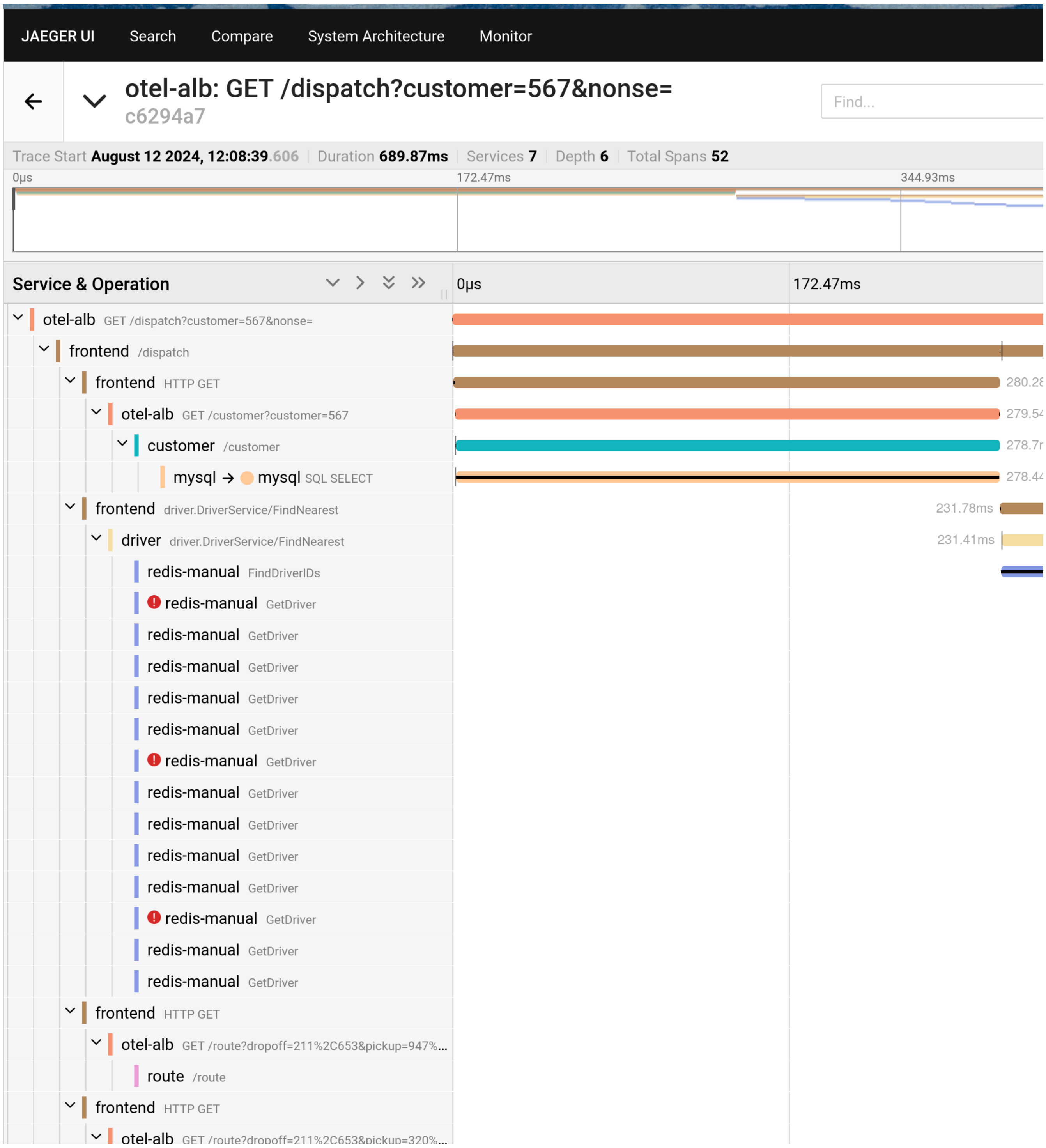
Sort: Most Recent

Compare traces by selecting result items

otel-alb: GET /dispatch?customer=567&nonse= c6294a7 689.87ms

52 Spans 3 Errors customer (1) driver (1) frontend (13) mysql (1) otel-alb (12) redis-manual (14) route (10)

Today 12:08:39 pm
a few seconds ago


JAEGER UI Search Compare System Architecture Monitor

otel-alb: GET /dispatch?customer=567&nonse=
c6294a7

Trace Start **August 12 2024, 12:08:39**.606 | Duration **689.87ms** | Services **7** | Depth **6** | Total Spans **52**

Service & Operation	
otel-alb GET /dispatch?customer=567&nonse=	
frontend /dispatch	
frontend HTTP GET	280.28
otel-alb GET /customer?customer=567	279.54
customer /customer	278.7
mysql → mysql SQL SELECT	278.44
frontend driver.DriverService/FindNearest	231.78ms
driver driver.DriverService/FindNearest	231.41ms
redis-manual FindDriverIDs	
❗ redis-manual GetDriver	
redis-manual GetDriver	
redis-manual GetDriver	
redis-manual GetDriver	
redis-manual GetDriver	
❗ redis-manual GetDriver	
redis-manual GetDriver	
redis-manual GetDriver	
redis-manual GetDriver	
redis-manual GetDriver	
❗ redis-manual GetDriver	
redis-manual GetDriver	
redis-manual GetDriver	
frontend HTTP GET	
otel-alb GET /route?dropoff=211%2C653&pickup=947%...	
route /route	
frontend HTTP GET	
otel-alb GET /route?dropoff=211%2C653&pickup=320%...	

ALB Monitoring

TOC

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

Monitoring Metric	Description
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 have a response time less than or equal to this value.

Monitoring Metric	Description
	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.

CORS

TOC

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`.

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

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

```yaml
spec:
  serviceGroup:
    session_affinity_policy: "<algorithm>"  # rr | sip-hash | cookie | header | ewma
    session_affinity_attribute: "<attribute>"  # Required for cookie and header 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

URL Rewrite

TOC

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` .

Calico Network Supports WireGuard Encryption

Calico supports WireGuard encryption for both IPv4 and IPv6 traffic, which can be independently enabled via parameters in the FelixConfiguration resource.

TOC

Installation Status

Default Installation

Operating System	Kernel Version
Linux	5.6 and above are installed by default
Ubuntu 20.04	5.4.0-135-generic

Operating System	Kernel Version
Kylin Linux Advanced Server V10 - SP3	4.19.90-52.22.v2207.ky10.x86_64

Not Installed by Default

Operating System	Kernel Version
openEuler	4.18.0-147.5.2.13.h996.eulerosv2r10.x86_64
CentOS 7	3.10.0-1160.el7.x86_64
Redhat 8.7	4.18.0-425.3.1.el8.x86_64
Kylin Linux Advanced Server V10 - SP2	4.19.90-24.4.v2101.ky10.x86_64
Kylin Linux Advanced Server V10 - SP1	4.19.90-23.8.v2101.ky10.x86_64
Kylin Linux Advanced Server V10	4.19.90-11.ky10.x86_64

Terminology

Term	Explanation
wireguardEnabled	Enable encryption for IPv4 traffic over the IPv4 Underlay network.
wireguardEnabledV6	Enable encryption for IPv6 traffic over the IPv6 Underlay network.

Notes

1. When using the Calico network plugin, ensure that the `natOutgoing` parameter is set to `true` to support WireGuard encryption. By default, this parameter is correctly configured for the Calico subnet when creating the cluster, requiring no additional configuration.

2. WireGuard supports encryption for both IPv4 and IPv6 traffic; if you need to encrypt both types of traffic, configuration must be done separately. For detailed parameter configuration, refer to the Felix Configuration Documentation↗, configuring both `wireguardEnabled` and `wireguardEnabledV6` parameters.

3. If WireGuard is not installed by default, refer to the WireGuard Installation Guide↗ for manual installation, although there may be cases where manual installation of the WireGuard module fails.

4. Traffic between containers across nodes will be encrypted, including network traffic from one host to another; however, communication between Pods on the same node and traffic between a Pod and its host node will not be encrypted.

Prerequisites

- WireGuard must be installed on all nodes in the cluster beforehand. For details, refer to the WireGuard Installation Documentation↗. Nodes without WireGuard installed do not support encryption.

Procedure

1. Enable or disable IPv4 and IPv6 encryption.

   Note: The following commands must be executed in the CLI tool on the Master node where the node resides.

   - Enable IPv4 encryption only

```
kubectl patch felixconfiguration default --type='merge' -p '{"spec":
{"wireguardEnabled":true}}'
```

- Enable IPv6 encryption only

```
kubectl patch felixconfiguration default --type='merge' -p '{"spec":
{"wireguardEnabledV6":true}}'
```

- Enable both IPv4 and IPv6 encryption

```
kubectl patch felixconfiguration default --type='merge' -p '{"spec":
{"wireguardEnabled":true,"wireguardEnabledV6":true}}'
```

- Disable both IPv4 and IPv6 encryption
  - Method 1: Execute the command in the CLI tool to disable encryption.

    ```
    kubectl patch felixconfiguration default --type='merge' -p '{"spec":
    {"wireguardEnabled":false,"wireguardEnabledV6":false}}'
    ```

  - Method 2: Modify the felixconfiguration configuration file to disable encryption.

    1. Execute the following command to open the felixconfiguration configuration file.

       ```
       kubectl get felixconfiguration -o yaml default
       ```

    2. Set `wireguardEnabled` and `wireguardEnabledV6` parameters to false to disable WireGuard encryption.

```yaml
apiVersion: crd.projectcalico.org/v1
kind: FelixConfiguration
metadata:
  annotations:
    projectcalico.org/metadata: '{"uid":"f5facabd-8304-46d6-81c1-
f1816235b487","creationTimestamp":"2024-08-06T03:46:51Z"}'
  generation: 2
  name: default
  resourceVersion: "890216"
spec:
  bpfLogLevel: ""
  floatingIPs: Disabled
  logSeverityScreen: Info
  reportingInterval: 0s
  wireguardEnabled: false # Change to true to enable IPv4 encryption
  wireguardEnabledV6: false # Change to true to enable IPv6 encryption
```

2. After completing the Calico WireGuard encryption configuration, execute the following command to confirm the WireGuard encryption status. If both IPv4 and IPv6 encryption are enabled, the presence of `wireguardPublicKey` or `wireguardPublicKeyV6` under the `Status` field indicates successful activation; if both IPv4 and IPv6 encryption are disabled, these fields will not contain `wireguardPublicKey` or `wireguardPublicKeyV6`, indicating successful deactivation.

   ```
   calicoctl get node <NODE-NAME> -o yaml # Replace <NODE-NAME> with the name of the
   node.
   ```

   Output:

   ```
   Status:
       wireguardPublicKey: L/MUP9+Yxx/xxxxxxxxxxxx/xxxxxxxxxx =
   ```

Result Verification

This document uses IPv4 traffic verification as an example; IPv6 traffic verification is similar to IPv4 and will not be repeated here.

IPv4 Traffic Verification

1. After configuring WireGuard encryption, check the routing information, where traffic between nodes preferentially uses the wireguard.cali interface for message forwarding.

```
root@test:~# ip rule   # View current routing rules
     0: from all lookup local
     99:  not from all fwmark 0x100000/0x100000 lookup 1    # For all packets not
marked as 0x100000, use routing table 1 for routing lookup
     32766:  from all lookup main
     32767 :  from all lookup default

root@test:~# ip route show table 1    # Display routing entries for table 1.
    10.3.138.0 dev wireguard.cali scope link
    10.3.138.0/26 dev wireguard.cali scope link
    throw 10.3.231.192
    10.3.236.128 dev wireguard.cali scope link     # Traffic to reach IP address
10.3.236.128 will be sent through the wireguard.cali interface
    10.3.236.128/26 dev wireguard.cali scope link
    throw 10.10.10.124/30
    10.10.10.200/30 dev wireguard.cali scope link
    throw 10.10.20.124/30
    10.10.20.200/30 dev wireguard.cali scope link
    throw
    10.13.138.0 dev wireguard.cali scope link
    10.13.138.0/26 dev wireguard.cali scope link
    throw 10.13.231.192/26
    10.13.236.128 dev wireguard.cali scope link
    10.13.236.128/26 dev wireguard.cali scope link

root@test:~# ip r get 10.10.10.202    # Routing path from the current node to the
target IP address 10.10.10.202
    10.10.10.202 dev wireguard.cali table 1 src 10.10.10.127 uid 0  cache   # When
accessing the target IP address 10.10.10.202 from the current node, the packet will be
sent through the wireguard.cali interface, using routing table 1, and the source
address will be set to 10.10.10.127

root@test:~# ip route    # Show the main routing table
    default via 192.168.128.1 dev eth0 proto static
    10.3.138.0/26 via 10.3.138.0 dev vxlan.
    blackhole 10.3.231.193
    10.3.231.194
    10.3.231.195
    10.3.231.196
    10.3.231.197
    3.231.192/26 proto 80
    dev cali8dcd31cId00 scope link
    dev cali3012b5b29b scope link
    dev calibeefea2ff87 scope link
    dev cali2b27d5e4053 scope link
    dev cali1a35dbdd639 scope link
    calico on link
```

2. Capture packets on the node to observe cross-node traffic.

```
root@test:~# ip a s wireguard.cali    # View detailed information about the
wireguard.cali network interface
    30: wireguard.cali: <POINTOPOINT,NOARP,UP,LOWER_UP> mtu 1440 qdisc noqueue state
UNKNOWN group default qlen 1000
    link/none
    inet 10.10.10.127/32 scope global wireguard.cali   # The IP address assigned to
wireguard.cali interface is 10.10.10.127
    valid_lft forever preferred_lft forever

root@test:~# tcpdump -i wireguard.cali -nnve icmp   # Capture and display ICMP packets
through wireguard.cali
    tcpdump: listening on wireguard.cali, link-type RAW (Raw IP), capture size 262144
bytes
    08:58:36.987559 ip: (tos 0x0, ttl 63, id 29731, offset 0, flags [DF], proto ICMP
(1), length 84)
    10.10.10.125 > 10.10.10.202: ICMP echo request, id 1110, seq 0, length 64
    08:58:36.988683 ip: (tos 0x0, ttl 63, id 1800, offset 0, flags [none], proto ICMP
(1), length 84)
    10.10.10.202 > 10.10.10.125: ICMP echo reply, id 1110, seq 0, length 64
    2 packets captured
    2 packets received by filter
    0 packets dropped by kernel
```

3. Testing shows that IPv4 type traffic is forwarded via the wireguard.cali interface.

Kube-OVN Overlay Network Supports IPsec Encryption

This document provides a detailed guide on enabling and disabling IPsec encrypted tunnel traffic in the Kube-OVN Overlay network. Since OVN tunnel traffic is transmitted through physical routers and switches, which may be located in untrusted public networks or at risk of attacks, enabling IPsec encryption can effectively prevent traffic data from being monitored and tampered with.

TOC

Terminology

Term	Explanation
IPsec	A protocol and technology used to protect and validate data transmitted over the internet. It provides secure communication at the IP layer and is primarily used to create virtual private networks (VPNs) and protect the transmission of IP packets. IPsec ensures data security primarily through the following methods:

Term	Explanation
	• **Data Encryption**: Through encryption technology, IPsec can ensure that data is not stolen or altered during transmission. Common encryption algorithms include AES, 3DES, etc. • **Data Integrity Check**: IPsec uses hash functions (such as SHA-1, SHA-256) to verify the integrity of data, ensuring that data has not been modified during transmission. • **Authentication**: IPsec can verify the identity of both parties involved in communication using various methods (such as pre-shared keys, digital certificates) to prevent unauthorized access. • **Key Management**: IPsec uses the Internet Key Exchange (IKE) protocol to negotiate and manage encryption keys.

Notes

- Enabling IPsec may cause a few seconds of network interruption.
- If the kernel version is 3.10.0-1160.el7.x86_64, enabling the IPsec feature of Kube-OVN may encounter compatibility issues.

Prerequisites

Please execute the following command to check whether the current operating system kernel supports IPsec-related modules. If the output shows that all XFRM-related modules are `y` or `m`, it indicates support for IPsec.

```
cat /boot/config-$(uname -r) | grep CONFIG_XFRM
```

Output:

```
CONFIG_XFRM_ALGO=y
CONFIG_XFRM_USER=y
CONFIG_XFRM_SUB_POLICY=y
CONFIG_XFRM_MIGRATE=y
CONFIG_XFRM_STATISTICS=y
CONFIG_XFRM_IPCOMP=m
```

Procedure

Note: Unless otherwise specified, the following commands must be executed in the CLI tool on the cluster Master node.

Enable IPsec

1. Modify the configuration file of kube-ovn-controller.

   1. Execute the following command to edit the YAML configuration file of kube-ovn-controller.

      ```
      kubectl edit deploy kube-ovn-controller -n kube-system
      ```

   2. Modify the specified fields according to the following instructions.

      ```
      spec:
        template:
          spec:
            containers:
              - args:
                  - --enable-ovn-ipsec=true # Add this field
                securityContext:
                  runAsUser: 0 # Change the value to 0
      ```

      Field explanations:

- **spec.template.spec.containers[0].args**: Add `- --enable-ovn-ipsec=true` under this field.
- **spec.template.spec.containers[0].securityContext.runAsUser**: Change the value of this field to 0.

3. Save the changes.

2. Modify the kube-ovn-cni configuration file.

1. Execute the following command to edit the YAML configuration file of kube-ovn-cni.

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

2. Modify the specified fields according to the following instructions.

```
spec:
  template:
    spec:
      containers:
        - args:
            - --enable-ovn-ipsec=true # Add this field
          volumeMounts: # Add mount path, mount the volume named ovs-ipsec-keys to the container
            - mountPath: /etc/ovs_ipsec_keys
              name: ovs-ipsec-keys
      volumes: # Add a volume named ovs-ipsec-keys of type hostPath
        - name: ovs-ipsec-keys
          hostPath:
            path: /etc/origin/ovs_ipsec_keys
```

Field explanations:

- **spec.template.spec.containers[0].args**: Add `- --enable-ovn-ipsec=true` under this field.
- **spec.template.spec.containers[0].volumeMounts**: Add the mount path and mount the volume named ovs-ipsec-keys to the container.
- **spec.template.spec.volumes**: Add a volume named ovs-ipsec-keys of type hostPath under this field.

3. Save the changes.

3. Verify whether the feature has been successfully enabled.

   1. Execute the following command to enter the kube-ovn-cni Pod.

      ```
      kubectl exec -it -n kube-system $(kubectl get pods -n kube-system -l app=kube-ovn-cni -o=jsonpath='{.items[0].metadata.name}') -- /bin/bash
      ```

   2. Execute the following command to check the number of Security Associations connections. If there are (number of nodes - 1) up, it indicates a successful enablement.

      ```
      ipsec status | grep "Security"
      ```

      Output:

      ```
      Security Associations (2 up, 0 connecting):  # Since there are 3 nodes in this cluster, you can see that the number of connections is 2 up
      ```

Disable IPsec

1. Modify the configuration file of kube-ovn-controller.

   1. Execute the following command to edit the YAML configuration file of kube-ovn-controller.

      ```
      kubectl edit deploy kube-ovn-controller -n kube-system
      ```

   2. Modify the specified fields according to the following instructions.

```yaml
spec:
  template:
    spec:
      containers:
        - args:
            - --enable-ovn-ipsec=false # Change to false
          securityContext:
            runAsUser: 65534 # Change the value to 65534
```

Field explanations:

- **spec.template.spec.containers[0].args**: Change the value of this field `enable-ovn-ipsec` to false.
- **spec.template.spec.containers[0].securityContext.runAsUser**: Change the value of this field to 65534.

3. Save the changes.

2. Modify the kube-ovn-cni configuration file.

1. Execute the following command to edit the YAML configuration file of kube-ovn-cni.

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

2. Modify the specified fields according to the following instructions.

- Configuration before modification

```yaml
spec:
  template:
    spec:
      containers:
        - args:
            - --enable-ovn-ipsec=true # Change to false
          volumeMounts: # Remove the mount path named ovs-ipsec-keys
            - mountPath: /etc/ovs_ipsec_keys
              name: ovs-ipsec-keys
      volumes: # Remove the volume named ovs-ipsec-keys, type hostPath
        - name: ovs-ipsec-keys
          hostPath:
            path: /etc/origin/ovs_ipsec_keys
```

Field explanations:

- **spec.template.spec.containers[0].args**: Change the value of this field `enable-ovn-ipsec` to false.
- **spec.template.spec.containers[0].volumeMounts**: Remove the mount path named ovs-ipsec-keys under this field.
- **spec.template.spec.volumes**: Remove the volume named ovs-ipsec-keys, type hostPath under this field.

- Configuration after modification

```yaml
spec:
  template:
    spec:
      containers:
        - args:
            - --enable-ovn-ipsec=false
          volumeMounts:
      volumes:
```

3. Save the changes.

3. Verify whether the feature has been successfully disabled.

1. Execute the following command to enter the kube-ovn-cni Pod.

```
kubectl exec -it -n kube-system $(kubectl get pods -n kube-system -l app=kube-ovn-
cni -o=jsonpath='{.items[0].metadata.name}') -- /bin/bash
```

2. Execute the following command to check the connection status. If there is no output, it indicates successful disabling.

```
ipsec status
```

Menu

Trouble Shooting

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
  ```

Menu

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 for 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.
```