Menu

Сетевые взаимодействия

Введение

Введение

Ограничения использования

Архитектура

Понимание Kube-OVN

Компоненты upstream OVN/OVS

Основной контроллер и агент

Инструменты мониторинга, эксплуатации и расширения

Понимание ALB

Основные компоненты

Быстрый старт

Взаимосвязь между ALB, ALB Instance, Frontend/FT, Rule, Ingress и Project

ALB Leader

Дополнительные ресурсы:

Понимание MetalLB

Терминология

Принципы высокой доступности в MetalLB

Алгоритм выбора узлов-хозяев VIP

Дополнительные ресурсы

Основные понятия

Совместимость ALB с аннотациями Ingress-NGINX

Основные понятия

Поддерживаемые аннотации ingress-nginx

Сравнение Service, Ingress, Gateway API и ALB Rule

Для L4 (TCP/UDP) трафика

Для L7 (HTTP/HTTPS) трафика

GatewayAPI

Через ALB

Руководства

Создание сервисов

Создание Ingress

Создание доменного имени

Создание сертификатов

Создание пула внешних IP-адресов

Создание BGP-пиров

Настройка подсетей

Настройка сетевых политик

Создание Admin Network Policies

Настройка сети Kube-OVN для поддержки нескольких сетевых интерфейсов Pod (Alpha)

Настройка сетевых политик кластера

Настройка Egress Gateway

Наблюдаемость сети

О DeepFlow

Установка DeepFlow

Дополнительные ресурсы

Настройка правил ALB

Введение

Действия

Backend

Создание правила

HTTPS

Ingress

Межкластерное соединение (Alpha)

Поддерживает настройку межкластерного соединения между кластерами с сетевым режимом Kube-OVN, чтобы обеспечить доступ подов между кластерами.

Предварительные требования

Построен контроллер подключения Multi-node Kube-OVN

Развертывание контроллера межкластерного соединения в глобальном кластере

Присоединение к межкластерному соединению

Соответствующие операции

Endpoint Health Checker

Overview

Key Features

Installation

How It Works

Uninstallation

NodeLocal DNSCache

Overview

Key Features

Important Notes

Installation

How It Works

Configuration

Как сделать

Подготовка физической сети Kube-OVN Underlay

Инструкции по использованию

Объяснение терминологии

Требования к среде

Пример конфигурации

Soft Data Center LB Solution (Alpha)

Предварительные требования

Процедура

Проверка

Автоматическое взаимное подключение подсетей Underlay и Overlay

Процедура

Установка Ingress-Nginx через Cluster Plugin

Overview

Installation

Configuration Management

Performance Tuning

Important Notes

Задачи для Ingress-Nginx

Предварительные требования

Максимальное количество соединений

Таймаут

Сессии с сохранением состояния (Sticky Session)

Модификация заголовков

Перезапись URL

HSTS (HTTP Strict Transport Security)

Ограничение скорости (Rate Limiting)

WAF

Управление заголовком Forward

HTTPS

ALB

Calico Network поддерживает шифрование WireGuard

Статус установки

Терминология

Примечания

Требования

Процедура

Проверка результата

Kube-OVN Overlay Network поддерживает шифрование IPsec

Терминология

Примечания

Предварительные требования

Процедура

Устранение неполадок

Как решить проблемы межузловой коммуникации в ARM-средах?

Определение причины ошибки

Введение

Сетевая инфраструктура контейнеров — это комплексное сетевое решение, разработанное для облачно-нативных приложений, обеспечивающее беспрепятственную коммуникацию east-west внутри кластеров и эффективное управление трафиком north-south между внешними сетями, при этом предоставляющее необходимые сетевые функции. Она состоит из следующих основных компонентов:

- Container Network Interfaces (CNI) для управления трафиком east-west внутри кластера.
- Ingress Gateway Controller ALB для управления HTTPS ingress трафиком.
- MetalLB для обработки сервисов типа LoadBalancer.
- Кроме того, она предоставляет надежные функции сетевой безопасности и шифрования для обеспечения защищенной коммуникации.

Содержание

Ограничения использования

Несмотря на широкие возможности сетевой инфраструктуры контейнеров, следует учитывать следующие ограничения:

- **Требование к подлежащей сети**

  Некоторые возможности подлежащей сети, такие как Kube-OVN Underlay Subnet, Egress IP и MetalLB, требуют поддержки L2 сети на нижнем уровне. Эти функции не

могут использоваться в публичных облачных провайдерах и некоторых виртуализированных средах, таких как AWS и GCP.

Благодаря универсальному дизайну и комплексному набору функций, сетевая инфраструктура контейнеров позволяет организациям создавать, масштабировать и управлять безопасными, надежными и высокопроизводительными контейнеризированными приложениями.

Menu

Архитектура

Понимание Kube-OVN

Компоненты upstream OVN/OVS

Основной контроллер и агент

Инструменты мониторинга, эксплуатации и расширения

Понимание ALB

Основные компоненты

Быстрый старт

Взаимосвязь между ALB, ALB Instance, Frontend/FT, Rule, Ingress и Project

ALB Leader

Дополнительные ресурсы:

Понимание MetalLB

Терминология

Принципы высокой доступности в MetalLB

Алгоритм выбора узлов-хозяев VIP

Дополнительные ресурсы

Понимание Kube-OVN

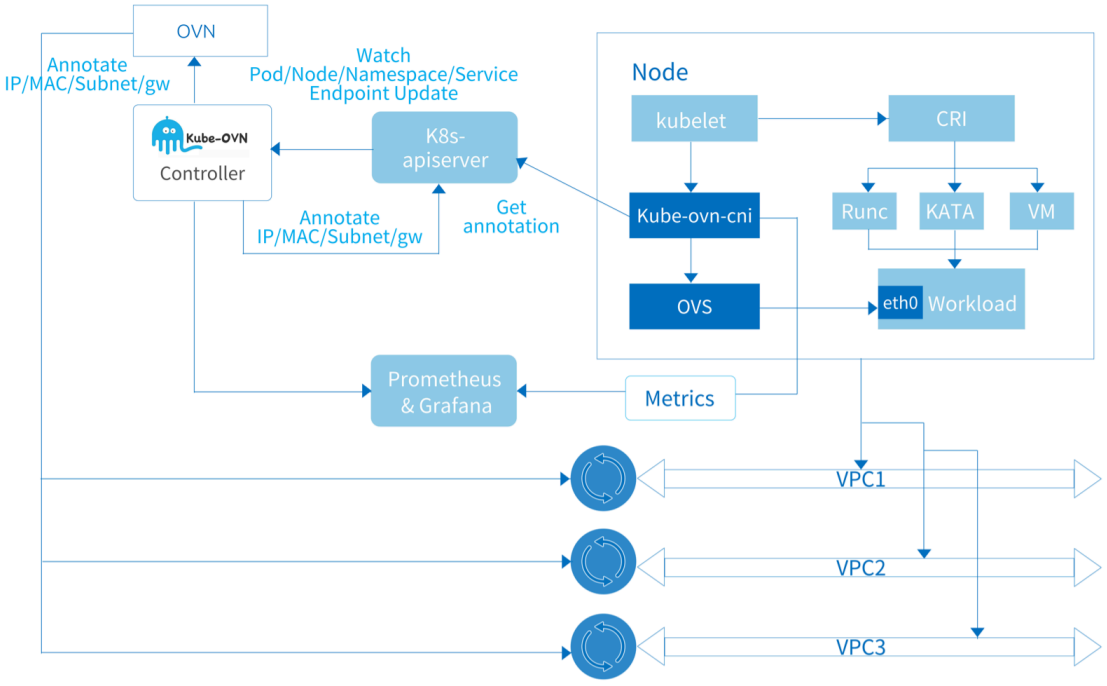
В этом документе описывается общая архитектура Kube-OVN, функциональность каждого компонента и их взаимодействие.

В целом, Kube-OVN служит мостом между Kubernetes и OVN, объединяя проверенные SDN с Cloud Native. Это означает, что Kube-OVN не только реализует сетевые спецификации в Kubernetes, такие как CNI, Service и NetworkPolicy, но и приносит множество возможностей SDN в облачную нативную среду, таких как логические коммутаторы, логические маршрутизаторы, VPC, шлюзы, QoS, ACL и зеркалирование трафика.

Kube-OVN также поддерживает хорошую открытость для интеграции с множеством технологических решений, таких как Cilium, Submariner, Prometheus, KubeVirt и др.

Компоненты Kube-OVN можно условно разделить на три категории.

- Компоненты upstream OVN/OVS.
- Основной контроллер и агент.
- Инструменты мониторинга, эксплуатации и расширения.



Содержание

Компоненты upstream OVN/OVS

Данный тип компонентов происходит из сообщества OVN/OVS с конкретными модификациями для сценариев использования Kube-OVN. OVN/OVS — это зрелая SDN-система для управления виртуальными машинами и контейнерами, и мы настоятельно рекомендуем пользователям, заинтересованным в реализации Kube-OVN, сначала ознакомиться с ovn-architecture(7)↗, чтобы понять, что такое OVN и как с ним интегрироваться. Kube-OVN использует northbound-интерфейс OVN для создания и координации виртуальных сетей и отображения сетевых концепций в Kubernetes.

Все компоненты, связанные с OVN/OVS, упакованы в образы и готовы к запуску в Kubernetes.

ovn-central

Deployment `ovn-central` запускает компоненты контрольной плоскости OVN, включая `ovn-nb`, `ovn-sb` и `ovn-northd`.

- `ovn-nb`: Сохраняет конфигурацию виртуальной сети и предоставляет API для управления виртуальной сетью. `kube-ovn-controller` в основном взаимодействует с `ovn-nb` для настройки виртуальной сети.
- `ovn-sb`: Хранит таблицу логических потоков, сгенерированную из логической сети `ovn-nb`, а также фактическое состояние физической сети каждого узла.
- `ovn-northd`: преобразует виртуальную сеть из `ovn-nb` в таблицу логических потоков в `ovn-sb`.

Несколько экземпляров `ovn-central` синхронизируют данные через протокол Raft для обеспечения высокой доступности.

ovs-ovn

`ovs-ovn` запускается как DaemonSet на каждом узле, внутри Pod работают `openvswitch`, `ovsdb` и `ovn-controller`. Эти компоненты выступают агентами для `ovn-central`, преобразуя таблицы логических потоков в реальные сетевые конфигурации.

Основной контроллер и агент

Эта часть является ядром Kube-OVN, служит мостом между OVN и Kubernetes, связывая две системы и переводя сетевые концепции между ними. Большинство основных функций реализованы в этих компонентах.

kube-ovn-controller

Этот компонент выполняет трансляцию всех ресурсов внутри Kubernetes в ресурсы OVN и выступает в роли контрольной плоскости всей системы Kube-OVN. `kube-ovn-controller` слушает события по всем ресурсам, связанным с сетевой функциональностью, и обновляет логическую сеть внутри OVN на основе изменений ресурсов. Основные ресурсы, за которыми ведется наблюдение, включают:

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

На примере события создания Pod: `kube-ovn-controller` слушает событие создания Pod, выделяет адрес с помощью встроенной функции IPAM в памяти, и вызывает `ovn-`

`central` для создания логических портов, статических маршрутов и возможных правил ACL. Далее `kube-ovn-controller` записывает назначенный адрес и информацию о подсети, такую как CIDR, шлюз, маршрут и т.д., в аннотацию Pod. Эту аннотацию затем читает `kube-ovn-cni` и использует для настройки локальной сети.

kube-ovn-cni

Этот компонент запускается на каждом узле как DaemonSet, реализует интерфейс CNI и управляет локальным OVS для настройки локальной сети.

DaemonSet копирует бинарный файл `kube-ovn` на каждую машину как инструмент взаимодействия между `kubelet` и `kube-ovn-cni`. Этот бинарный файл отправляет соответствующий CNI-запрос в `kube-ovn-cni` для дальнейших операций. По умолчанию бинарный файл копируется в каталог `/opt/cni/bin`.

`kube-ovn-cni` настраивает конкретную сеть для выполнения соответствующих операций с трафиком, основные задачи включают:

1. Настройка `ovn-controller` и `vswitchd`.
2. Обработка запросов CNI Add/Del:
    2.1. Создание или удаление пары veth и привязка или отвязка к портам OVS.
    2.2. Настройка портов OVS.
    2.3. Обновление правил iptables/ipset/route на хосте.
3. Динамическое обновление QoS сети.
4. Создание и настройка NIC `ovn0` для соединения сети контейнеров и сети хоста.
5. Настройка NIC хоста для реализации Vlan/Underlay/EIP.
6. Динамическая настройка межкластерных шлюзов.

Инструменты мониторинга, эксплуатации и расширения

Эти компоненты предоставляют средства мониторинга, диагностики, инструменты эксплуатации и внешние интерфейсы для расширения основных сетевых возможностей Kube-OVN и упрощения повседневных операций и обслуживания.

kube-ovn-speaker

Этот компонент — DaemonSet, работающий на узлах с определенной меткой, который публикует маршруты во внешний мир, позволяя внешнему доступу к контейнерам напрямую через IP Pod.

kube-ovn-pinger

Этот компонент — DaemonSet, работающий на каждом узле, собирает информацию о состоянии OVS, качестве сети узла, задержках в сети и т.д.

kube-ovn-monitor

Этот компонент собирает информацию о состоянии OVN и метрики мониторинга.

kubectl-ko

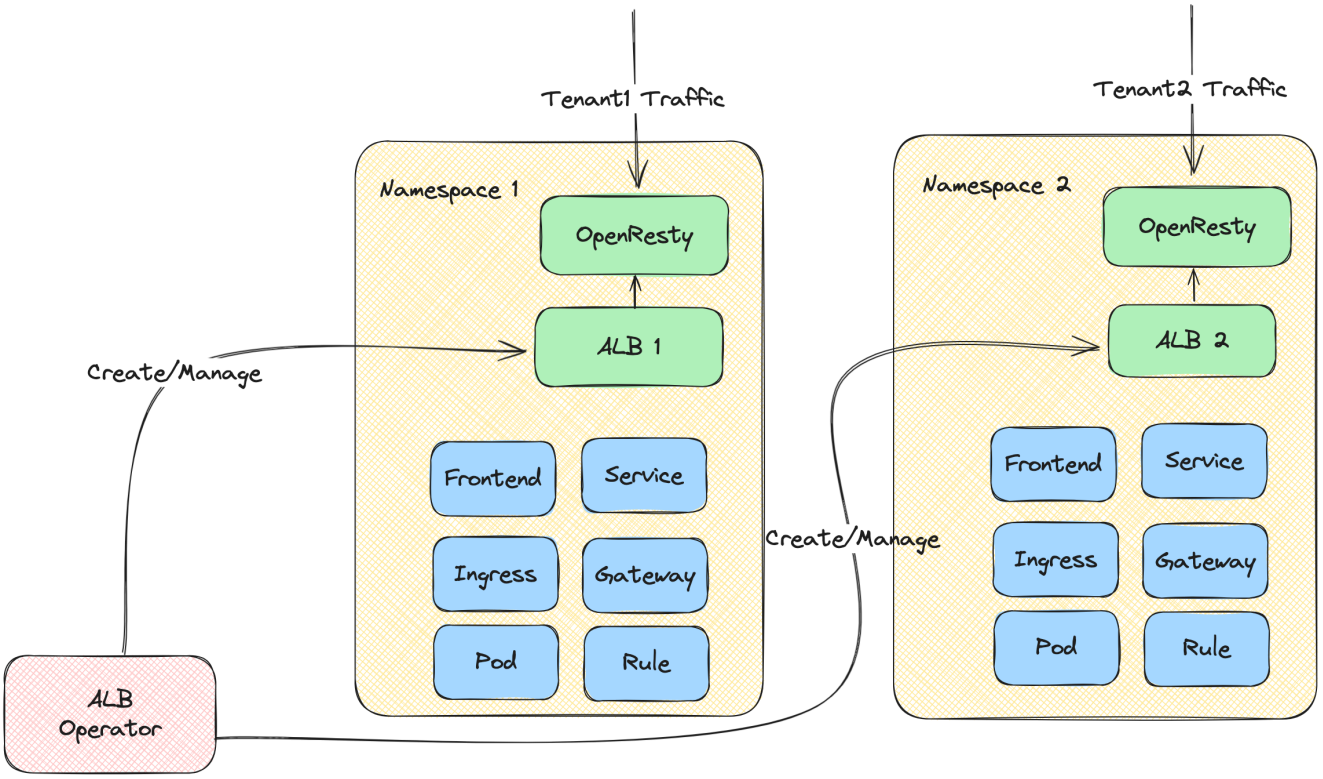
Этот компонент — плагин kubectl, который позволяет быстро выполнять распространённые операции.

Понимание ALB

ALB (Another Load Balancer) — это Kubernetes Gateway, основанный на OpenResty, с многолетним опытом эксплуатации в продакшене от Alauda.

Содержание

Основные компоненты



- **ALB Operator**: оператор, управляющий жизненным циклом экземпляров ALB. Он отслеживает CR ALB и создает/обновляет экземпляры для разных арендаторов.
- **ALB Instance**: экземпляр ALB включает OpenResty, выступающий в роли data plane, и контроллер на Go в роли control plane. Контроллер на Go следит за различными CR (Ingress, Gateway, Rule и др.) и преобразует их в специфичные для ALB DSL-правила. OpenResty затем использует эти DSL-правила для сопоставления и обработки входящих запросов.

Быстрый старт

Развертывание ALB Operator

1. Создайте кластер.
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
```

Развертывание экземпляра ALB

```
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"  # IP-адрес ноды, на которой развернут alb
    type: "nginx"
    config:
        networkMode: host
        loadbalancerName: alb-demo
        projects:
        - ALL_ALL
        replicas: 1
EOF
```

Запуск демонстрационного приложения

```
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
```

Теперь вы можете получить доступ к приложению через `curl http://${ip}`

Взаимосвязь между ALB, ALB Instance, Frontend/FT, Rule, Ingress и Project

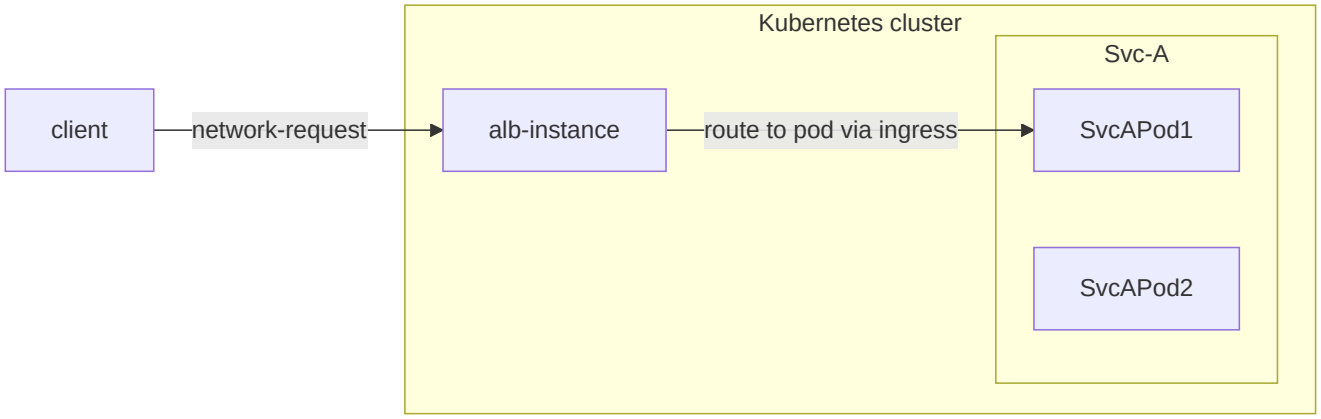
LoadBalancer — ключевой компонент современных облачно-нативных архитектур, выступающий в роли интеллектуального маршрутизатора трафика и балансировщика нагрузки.

Чтобы понять, как работает ALB в Kubernetes-кластере, необходимо разобраться в нескольких основных понятиях и их взаимосвязях:

- Сам ALB
- Frontend (FT)
- Правила (Rules)
- Ресурсы Ingress
- Проекты (Projects)

Эти компоненты работают совместно, обеспечивая гибкое и мощное управление трафиком.

Ниже описано, как эти концепции взаимодействуют в цепочке обработки запроса. Подробные описания каждого понятия приведены в отдельных статьях.



В цепочке вызова запроса:

1. Клиент отправляет HTTP/HTTPS/другой протокол запрос, который в конечном итоге **попадает на pod ALB**, и этот pod (экземпляр ALB) начинает обрабатывать запрос.
2. Экземпляр ALB находит правило, которое может соответствовать этому запросу.
3. При необходимости модифицирует/перенаправляет/переписывает запрос на основе правила.
4. Находит и выбирает IP одного из pod-ов сервиса, указанного в правиле, и пересылает запрос на этот pod.

Ingress

Ingress — ресурс в Kubernetes, используемый для описания, какой запрос должен быть направлен к какому сервису.

Ingress Controller

Программа, которая понимает ресурс Ingress и проксирует запросы к сервису.

ALB

ALB — это Ingress controller.

В Kubernetes-кластере мы используем ресурс `alb2` для управления ALB. Вы можете использовать `kubectl get alb2 -A`, чтобы просмотреть все ALB в кластере.

ALB создаются пользователями вручную. Каждый ALB имеет свой собственный IngressClass. При создании Ingress можно использовать поле `.spec.ingressClassName`, чтобы указать, какой Ingress controller должен обрабатывать этот Ingress.

ALB Instance

ALB также представляет собой Deployment (набор pod-ов), работающий в кластере. Каждый pod называется экземпляром ALB.

Каждый экземпляр ALB обрабатывает запросы независимо, но все экземпляры разделяют Frontend (FT), Rule и другие конфигурации, принадлежащие одному ALB.

ALB-Operator

ALB-Operator — компонент по умолчанию, развернутый в кластере, оператор для ALB. Он создает/обновляет/удаляет Deployment и другие связанные ресурсы для каждого ALB в соответствии с ресурсом ALB.

Frontend (сокращенно: FT)

FT — ресурс, определяемый самим ALB. Он используется для представления портов прослушивания экземпляра ALB.

FT может быть создан ALB-Leader или пользователем вручную.

Случаи создания FT ALB-Leader:

1. Если у Ingress есть сертификат, создается FT 443 (HTTPS).
2. Если у Ingress нет сертификата, создается FT 80 (HTTP).

RULE

RULE — ресурс, определяемый самим ALB. Он выполняет ту же роль, что и Ingress, но более конкретен. RULE уникально связан с FT.

RULE может быть создан ALB-Leader или пользователем вручную.

Случаи создания RULE ALB-Leader:

1. Синхронизация Ingress в RULE.

ALB Leader

В нескольких экземплярах ALB один выбирается лидером. Лидер отвечает за:

1. Преобразование Ingress в Rules. Для каждого пути в Ingress создается Rule.
2. Создание FT, необходимых для Ingress. Например, если у Ingress есть сертификат, создается FT 443 (HTTPS), если нет — FT 80 (HTTP).

Project

С точки зрения ALB, Project — это набор namespace-ов.

Вы можете настроить один или несколько Projects в ALB. Когда ALB Leader преобразует Ingress в Rules, он игнорирует Ingress в namespace-ах, не входящих в Project.

Дополнительные ресурсы:

- Настройка Load Balancer

Понимание MetalLB

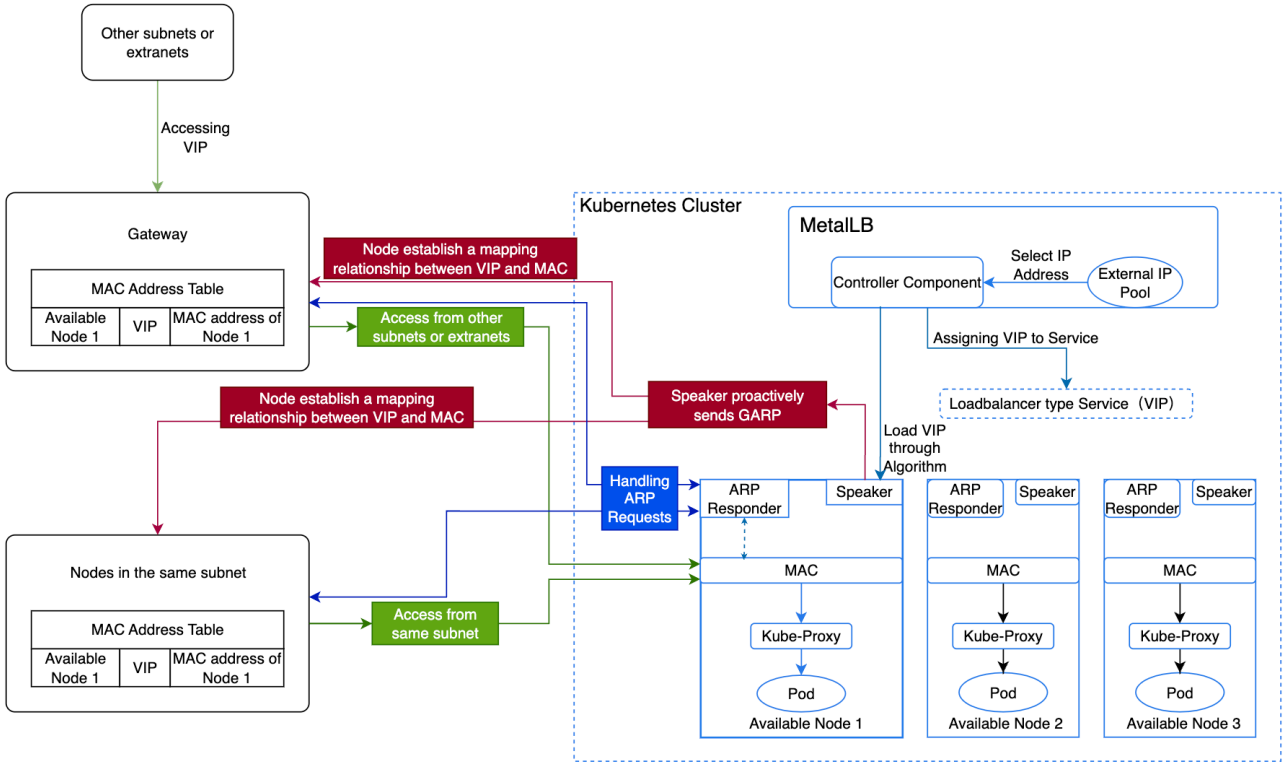
Содержание

Терминология

Термин	Описание
VIP	Виртуальный IP-адрес (VIP) — это IP-адрес, назначаемый MetalLB для внутренней маршрутизации типа LoadBalancer, обеспечивающий единый точку доступа для внешнего трафика к сервисам внутри кластера.
ARP	Протокол разрешения адресов (ARP) используется для сопоставления IP-адресов сетевого уровня с MAC-адресами канального уровня.
GARP	Gratuitous ARP (GARP) — это специальный ARP-запрос, используемый для информирования других узлов в сети о привязке IP-адреса к MAC-адресу. В отличие от обычных ARP-запросов, GARP не ожидает ответов, а активно рассылает информацию по сети.

Термин	Описание
ARP Responder	Компонент MetalLB, отвечающий на ARP-запросы, сопоставляя VIP с MAC-адресом узла. Когда узлу необходимо связаться с VIP, он отправляет ARP-запросы для получения MAC-адреса, соответствующего VIP. Каждый доступный узел имеет ARP Responder, который отвечает на эти запросы, сопоставляя VIP с MAC-адресом узла.
Controller	Компонент MetalLB, который динамически выделяет VIP из пула внешних адресов для внутренней маршрутизации типа LoadBalancer. Controller отслеживает события создания и удаления внутренних маршрутов в кластере для выделения или освобождения VIP по мере необходимости.
Speaker	Компонент MetalLB, который на основе политик или алгоритмов определяет, должны ли узлы размещать VIP и отправлять GARP. Он обеспечивает определённый уровень баланса между узлами, и когда узел становится недоступен, другие узлы могут взять на себя VIP и отправить GARP, обеспечивая тем самым высокую доступность.

Принципы высокой доступности в MetalLB



По умолчанию платформа использует ARP-режим MetalLB, а конкретный процесс реализации и принципы следующие:

- Компонент Controller MetalLB выбирает IP-адрес из пула внешних адресов и выделяет его для внутренней маршрутизации типа LoadBalancer в качестве VIP.
- MetalLB выбирает доступный узел в качестве лидера для размещения VIP на основе алгоритма, который затем перенаправляет трафик.
- Компонент Speaker на этом узле активно отправляет GARP, устанавливая связь между VIP и MAC-адресом на всех узлах.
  - Узлы в одной подсети, узнав сопоставление между VIP и MAC-адресом доступного узла, будут напрямую взаимодействовать с этим узлом при обращении к VIP.
  - Узлы в разных подсетях сначала направляют трафик на шлюз своей подсети, который затем перенаправляет трафик на узел, размещающий VIP.
- При сбое этого узла MetalLB выбирает другого лидера для размещения VIP и отправляет GARP для обновления MAC-адреса IP-сервиса, обеспечивая тем самым высокую доступность.
- По достижении узла Kube-Proxy перенаправляет трафик соответствующему Pod.

Алгоритм выбора узлов-хозяев VIP

Выбор «лидера» (узла, который будет рекламировать IP) для конкретного IP-адреса балансировщика нагрузки является stateless и работает следующим образом:

- каждый speaker собирает список потенциальных объявителей данного IP, учитывая активных speakers, политику внешнего трафика, активные endpoints, селекторы узлов и другие параметры.
- каждый speaker выполняет одинаковый расчёт: получает отсортированный список хэшей элементов «node+VIP» и объявляет сервис, если он является первым элементом списка.

Это устраняет необходимость хранить память о том, какой speaker отвечает за рекламу данного IP.

Формула расчёта

Формула: **Количество пулов внешних адресов = ceil(n-vip / n-node)**, где ceil — округление вверх.

Примечание: При использовании виртуальных машин количество виртуальных машин = Количество пулов внешних адресов * n. Здесь n должно быть больше 2, с допущением отказа не более одного узла.

- n-vip: количество VIP.
- n-node: количество VIP, которое может обслуживать один узел.

Пример применения

Если в компании 10 VIP, и каждый доступный узел может обслуживать 5 VIP, при допущении отказа одного узла, как компании спланировать количество пулов внешних адресов и доступных узлов?

Анализ:

Требуется всего два пула внешних адресов и четыре доступных узла.

- Каждый доступный узел может обслуживать максимум 5 VIP, значит один пул внешних адресов может вместить 5 VIP, следовательно, для 10 VIP нужны два пула внешних адресов.
- При допущении отказа одного узла каждый пул адресов должен включать один узел с VIP и один резервный узел, что даёт два доступных узла для каждого из двух пулов внешних адресов.

Дополнительные ресурсы

- Создание пула внешних IP-адресов
- Создание BGP-пиров

Menu

Основные понятия

Совместимость ALB с аннотациями Ingress-NGINX

Основные понятия

Поддерживаемые аннотации ingress-nginx

Сравнение Service, Ingress, Gateway API и ALB Rule

Для L4 (TCP/UDP) трафика

Для L7 (HTTP/HTTPS) трафика

GatewayAPI

Через ALB

Совместимость ALB с аннотациями Ingress-NGINX

Содержание

Основные понятия

ingress-nginx — это широко используемый Ingress Controller в Kubernetes, который определяет множество аннотаций для реализации различных функций, выходящих за рамки официального определения ingress.

ALB поддерживает часть этих аннотаций.

Поддерживаемые аннотации ingress-nginx

Название	тип	Поддержка (v — поддерживается, x — не поддерживается, o — частично поддерживается или может быть реализовано через конфигурацию)
nginx.ingress.kubernetes.io/app-root	string	x
nginx.ingress.kubernetes.io/affinity	cookie	o ingress не поддерживает. В правиле alb можно настроить cookie hash
nginx.ingress.kubernetes.io/use-regex	bool	
nginx.ingress.kubernetes.io/affinity-mode	"balanced" или "persistent"	o ingress не поддерживает. В правиле alb можно настроить сохранение сессии
nginx.ingress.kubernetes.io/affinity-canary-behavior	"sticky" или "legacy"	o ingress не поддерживает. В правиле alb можно настроить сохранение сессии
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" или "digest"	v auth

Название	тип	Поддержка (v — поддерживается, x — не поддерживается, o — частично поддерживается или может быть реализовано через конфигурацию)
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" или "false"	x
nginx.ingress.kubernetes.io/auth-tls-match-cn	string	x
nginx.ingress.kubernetes.io/auth-url	string	v
nginx.ingress.kubernetes.io/auth-cache-key	string	x
nginx.ingress.kubernetes.io/auth-cache-duration	string	x
nginx.ingress.kubernetes.io/auth-keepalive	number	x
nginx.ingress.kubernetes.io/auth-keepalive-share-vars	"true" или "false"	x
nginx.ingress.kubernetes.io/auth-keepalive-requests	number	x

Название	тип	Поддержка (v — поддерживается, x — не поддерживается, o — частично поддерживается или может быть реализовано через конфигурацию)
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" или "false"	o auth
nginx.ingress.kubernetes.io/backend-protocol	string	v
nginx.ingress.kubernetes.io/canary	"true" или "false"	x
nginx.ingress.kubernetes.io/canary-by-header	string	x
nginx.ingress.kubernetes.io/canary-by-header-value	string	x
nginx.ingress.kubernetes.io/canary-by-header-pattern	string	x
nginx.ingress.kubernetes.io/canary-by-cookie	string	x
nginx.ingress.kubernetes.io/canary-weight	number	x

Название	тип	Поддержка (v — поддерживается, x — не поддерживается, o — частично поддерживается или может быть реализовано через конфигурацию)
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 можно использовать default-backend ingress
nginx.ingress.kubernetes.io/enable-cors	"true" или "false"	v
nginx.ingress.kubernetes.io/cors-allow-origin	string	v
nginx.ingress.kubernetes.io/cors-allow-methods	string	v
nginx.ingress.kubernetes.io/cors-allow-headers	string	v
nginx.ingress.kubernetes.io/cors-expose-headers	string	x

Название	тип	Поддержка (v — поддерживается, x — не поддерживается, o — частично поддерживается или может быть реализовано через конфигурацию)
nginx.ingress.kubernetes.io/cors-allow-credentials	"true" или "false"	x
nginx.ingress.kubernetes.io/cors-max-age	number	x
nginx.ingress.kubernetes.io/force-ssl-redirect	"true" или "false"	v redirect
nginx.ingress.kubernetes.io/from-to-www-redirect	"true" или "false"	x
nginx.ingress.kubernetes.io/http2-push-preload	"true" или "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
nginx.ingress.kubernetes.io/global-rate-limit-ignored-cidrs	string	x
nginx.ingress.kubernetes.io/permanent-redirect	string	v redirect

Название	тип	Поддержка (v — поддерживается, x — не поддерживается, o — частично поддерживается или может быть реализовано через конфигурацию)
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" или "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

Название	тип	Поддержка (v — поддерживается, x — не поддерживается, o — частично поддерживается или может быть реализовано через конфигурацию)
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" или "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

Название	тип	Поддержка (v — поддерживается, x — не поддерживается, o — частично поддерживается или может быть реализовано через конфигурацию)
nginx.ingress.kubernetes.io/enable-rewrite-log	"true" или "false"	x
nginx.ingress.kubernetes.io/rewrite-target	URI	v
nginx.ingress.kubernetes.io/satisfy	string	x
nginx.ingress.kubernetes.io/server-alias	string	x
nginx.ingress.kubernetes.io/server-snippet	string	x
nginx.ingress.kubernetes.io/service-upstream	"true" или "false"	x
nginx.ingress.kubernetes.io/session-cookie-change-on-failure	"true" или "false"	x
nginx.ingress.kubernetes.io/session-cookie-conditional-samesite-none	"true" или "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

Название	тип	Поддержка (v — поддерживается, x — не поддерживается, o — частично поддерживается или может быть реализовано через конфигурацию)
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" или "false"	v
nginx.ingress.kubernetes.io/ssl-passthrough	"true" или "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 можно достичь аналогичного эффекта через modsecurity

Название	тип	Поддержка (v — поддерживается, x — не поддерживается, o — частично поддерживается или может быть реализовано через конфигурацию)
nginx.ingress.kubernetes.io/whitelist-source-range	CIDR	o можно достичь аналогичного эффекта через 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
nginx.ingress.kubernetes.io/ssl-prefer-server-ciphers	"true" или "false"	x
nginx.ingress.kubernetes.io/connection-proxy-header	string	x
nginx.ingress.kubernetes.io/enable-access-log	"true" или "false"	o по умолчанию access_log включён, формат фиксирован
nginx.ingress.kubernetes.io/enable-opentelemetry	"true" или "false"	v otel

Название	тип	Поддержка (v — поддерживается, x — не поддерживается, o — частично поддерживается или может быть реализовано через конфигурацию)
nginx.ingress.kubernetes.io/opentelemetry-trust-incoming-span	"true" или "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

Сравнение Service, Ingress, Gateway API и ALB Rule

Платформа Alauda Container Platform поддерживает несколько спецификаций ingress-трафика в экосистеме Kubernetes.
В этом документе проводится их сравнение (Service, Ingress, Gateway API и ALB Rule), чтобы помочь пользователям сделать правильный выбор.

Содержание

Для L4 (TCP/UDP) трафика

Сервисы типа LoadBalancer, Gateway API и ALB Rules могут все обеспечивать внешний доступ к L4 трафику. Здесь мы рекомендуем использовать подход с Service типа LoadBalancer.
И Gateway API, и ALB Rules реализованы через ALB, который является прокси в пространстве пользователя, и их производительность значительно снижается при обработке L4 трафика по сравнению с Service типа LoadBalancer.

Для L7 (HTTP/HTTPS) трафика

Хотя Ingress, GatewayAPI и ALB Rules все могут обеспечивать внешний доступ к L7 трафику, они отличаются по возможностям и моделям изоляции.

Ingress

Ingress — это стандартная спецификация, принятая сообществом Kubernetes, и рекомендуется для использования по умолчанию.
Ingress обрабатывается экземплярами ALB, которые управляются администратором платформы.

GatewayAPI

GatewayAPI предоставляет более гибкий режим изоляции, однако он менее зрелый, чем Ingress.
Используя GatewayAPI, разработчик может создавать собственные изолированные экземпляры ALB для обработки правил GatewayAPI.
Поэтому, если вам нужно делегировать создание и управление экземплярами ALB разработчикам, следует выбрать GatewayAPI.

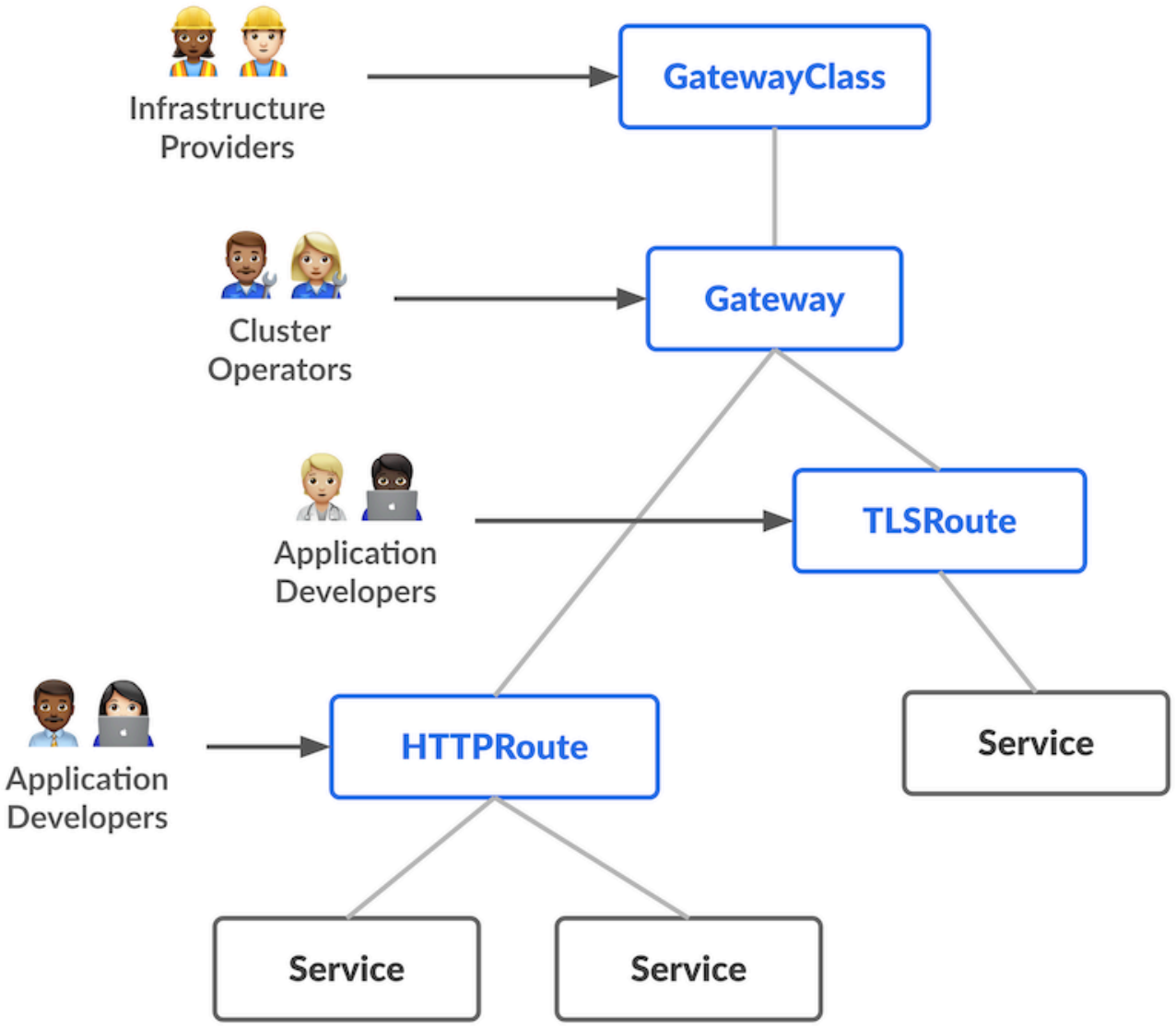
ALB Rule

ALB Rule (Load Balancer в UI) предоставляет наиболее гибкие правила сопоставления трафика и самые широкие возможности. Фактически, и Ingress, и GatewayAPI реализованы путем преобразования их в ALB Rules.
Однако ALB Rule сложнее, чем Ingress и GatewayAPI, и не является стандартным API сообщества. Поэтому мы рекомендуем использовать его только в тех случаях, когда Ingress и GatewayAPI не удовлетворяют вашим требованиям.

GatewayAPI

GatewayAPI ↗ — это официальный проект Kubernetes, ориентированный на маршрутизацию L4 и L7 в Kubernetes. Этот проект представляет собой следующее поколение API для Kubernetes Ingress, балансировки нагрузки и Service Mesh. С самого начала он был разработан как универсальный, выразительный и ориентированный на роли.

Общая модель ресурсов сосредоточена на 3 отдельных ролях и соответствующих ресурсах, которыми они должны управлять:



Содержание

Через ALB

ALB поддерживает GatewayAPI. Каждый ресурс Gateway сопоставляется с ресурсом ALB. Listeners и routes обрабатываются напрямую ALB; они не преобразуются в `Frontend` или `Rule`. Создание GatewayAPI Gateway через ALB

Menu

Руководства

Создание сервисов

Зачем нужен сервис

Пример сервиса типа ClusterIP:

Headless-сервисы

Создание сервиса через веб-консоль

Создание сервиса через CLI

Пример: Доступ к приложению внутри кластера

Пример: Доступ к приложению вне кластера

Пример: Сервис типа ExternalName

Аннотации для сервиса типа LoadBalancer

Создание Ingress

Метод реализации

Пример Ingress:

Создание Ingress через веб-консоль

Создание Ingress через CLI

Создание доменного имени

Пример ресурса Domain custom resource (CR)

Создание домена через веб-консоль

Создание домена с помощью CLI

Последующие действия

Дополнительные ресурсы

Создание сертификатов

Создание сертификата через веб-консоль

Создание пула внешних IP-адресов

Предварительные требования

Ограничения и условия

Развертывание плагина MetalLB

Пример ресурса custom resource (CR) IPAddressPool

Создание пула внешних IP-адресов через веб-консоль

Создание пула внешних IP-адресов через CLI

Просмотр политики оповещений

Создание BGP-пиров

Терминология

Предварительные требования

Пример пользовательского ресурса BGPPeer (CR)

Создание BGPPeer через веб-консоль

Создание BGPPeer через CLI

Настройка подсетей

Правила выделения IP

Сеть Calico

Сеть Kube-OVN

Управление подсетями

Настройка сетевых политик

Создание NetworkPolicy через веб-консоль

Создание NetworkPolicy с помощью CLI

Справка

Создание Admin Network Policies

Примечания

Создание AdminNetworkPolicy или BaselineAdminNetworkPolicy через веб-консоль

Создание AdminNetworkPolicy или BaselineAdminNetworkPolicy через CLI

Дополнительные ресурсы

Настройка сети Kube-OVN для поддержки нескольких сетевых интерфейсов Pod (Alpha)

Установка Multus CNI

Создание подсетей

Создание Pod с несколькими сетевыми интерфейсами

Проверка создания двух сетевых интерфейсов

Дополнительные возможности

Настройка сетевых политик кластера

Примечания

Процедура

Настройка Egress Gateway

Описание Egress Gateway

Примечания

Использование

Параметры конфигурации

Дополнительные ресурсы

Наблюдаемость сети

О DeepFlow

Установка DeepFlow

Дополнительные ресурсы

Настройка правил ALB

Введение

Действия

Backend

Создание правила

HTTPS

Ingress

Межкластерное соединение (Alpha)

Поддерживает настройку межкластерного соединения между кластерами с сетевым режимом Kube-OVN, чтобы обеспечить доступ подов между кластерами.

Предварительные требования

Построен контроллер подключения Multi-node Kube-OVN

Развертывание контроллера межкластерного соединения в глобальном кластере

Присоединение к межкластерному соединению

Соответствующие операции

Endpoint Health Checker

NodeLocal DNSCache

Создание сервисов

В Kubernetes сервис — это способ открыть сетевое приложение, работающее в виде одного или нескольких Pod в вашем кластере.

Содержание

Зачем нужен сервис

1. У Pod есть собственные IP, но:

- IP Pod нестабильны (меняются при пересоздании Pod).

- Прямой доступ к Pod становится ненадежным.

2. Сервис решает эту проблему, предоставляя:

- Стабильный IP и DNS-имя.
- Автоматическое балансирование нагрузки на соответствующие Pod.

Пример сервиса типа ClusterIP:

```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) Доступные значения type и их поведение: `ClusterIP`, `NodePort`, `LoadBalancer`, `ExternalName`

(2) Набор Pod, на которые нацелен сервис, обычно определяется селектором, который вы задаёте.

(3) Порт сервиса.

(4) Привязка `targetPort` сервиса к `containerPort` Pod. Также можно ссылаться на `port.name` внутри контейнера Pod.

Headless-сервисы

Иногда не требуется балансировка нагрузки и единый IP сервиса. В таком случае можно создать так называемые headless-сервисы:

```
spec:
  clusterIP: None
```

Headless-сервисы полезны, когда:

- Нужно обнаруживать отдельные IP Pod, а не только один IP сервиса.
- Требуются прямые подключения к каждому Pod (например, для баз данных типа Cassandra или StatefulSets).
- Используются StatefulSets, где каждый Pod должен иметь стабильное DNS-имя.

Создание сервиса через веб-консоль

1. Перейдите в **Container Platform**.
2. В левой панели навигации выберите **Network** > **Services**.
3. Нажмите **Create Service**.
4. Следуйте инструкциям для настройки соответствующих параметров.

Параметр	Описание
Virtual IP Address	Если включено, для сервиса будет выделен ClusterIP, который можно использовать для обнаружения сервиса внутри кластера. Если отключено, будет создан headless-сервис, который обычно используется для **StatefulSet**.
Type	• **ClusterIP**: Открывает сервис на внутреннем IP кластера. При выборе этого значения сервис доступен только внутри кластера.

Параметр	Описание
	• **NodePort**: Открывает сервис на IP каждого узла на статическом порту (NodePort). • **ExternalName**: Отображает сервис на содержимое поля externalName (например, на hostname api.foo.bar.example). • **LoadBalancer**: Открывает сервис снаружи с помощью внешнего балансировщика нагрузки. Kubernetes напрямую не предоставляет компонент балансировки нагрузки; необходимо предоставить его самостоятельно или интегрировать кластер с облачным провайдером.
Target Component	• **Workload**: Сервис будет перенаправлять запросы на **конкретную** нагрузку, которая соответствует меткам, например `project.cpaas.io/name: projectname` и `service.cpaas.io/name: deployment-name`. • **Virtualization**: Сервис будет перенаправлять запросы на **конкретную** виртуальную машину или группу виртуальных машин. • **Label Selector**: Сервис будет перенаправлять запросы на **определённый тип** нагрузки с указанными метками, например `environment: release`.
Port	Используется для настройки сопоставления портов сервиса. В следующем примере другие Pod внутри кластера могут обращаться к сервису по виртуальному IP (если включено) и TCP-порту *80*; запросы будут перенаправлены на внешний TCP-порт *6379* или *redis* Pod целевого компонента. • **Protocol**: Протокол, используемый сервисом, поддерживаются: `TCP`, `UDP`, `HTTP`, `HTTP2`, `HTTPS`, `gRPC`. • **Service Port**: Номер порта сервиса, открытого внутри кластера, то есть Port, например *80*.

Параметр	Описание
	• **Container Port**: Целевой номер порта (или имя), на который маппится порт сервиса, то есть targetPort, например *6379* или *redis*. • **Service Port Name**: Генерируется автоматически. Формат: `<protocol>-<service port>-<container port>`, например: *tcp-80-6379* или *tcp-80-redis*.
Session Affinity	Сессия с привязкой по IP-адресу источника (ClientIP). Если включено, все запросы с одного и того же IP будут направляться на один и тот же сервер при балансировке нагрузки, что гарантирует обработку запросов от одного клиента одним сервером.

5. Нажмите **Create**.

Создание сервиса через CLI

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

Создать сервис на основе существующего ресурса deployment `my-app`.

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

Пример: Доступ к приложению внутри кластера

```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. Примените этот YAML:

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

2. Запустите другой Pod:

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

3. Доступ к сервису `nginx-clusterip` из Pod `test-pod`:

```
wget -qO- http://nginx-clusterip
# или используя DNS-записи, созданные автоматически Kubernetes: <service-name>.
<namespace>.svc.cluster.local
wget -qO- http://nginx-clusterip.default.svc.cluster.local
```

Вы должны увидеть HTML-ответ с текстом вроде "Welcome to nginx!".

Пример: Доступ к приложению вне кластера

```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. Примените этот YAML:

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

2. Проверка Pod:

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

3. curl к сервису:

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

Вы должны увидеть HTML-ответ с текстом вроде "Welcome to nginx!".

Разумеется, можно также получить доступ к приложению снаружи кластера, создав сервис типа LoadBalancer.

Примечание: Пожалуйста, настройте сервис LoadBalancer заранее.

```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. Примените этот YAML:

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

2. Получите внешний IP-адрес:

```
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` — это адрес, по которому вы можете получить доступ из браузера.

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

Вы должны увидеть HTML-ответ с текстом вроде "Welcome to nginx!".

Если EXTERNAL-IP равен `pending`, значит сервис LoadBalancer в данный момент не развернут в кластере.

Пример: Сервис типа ExternalName

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

1. Примените этот YAML:

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

2. Попробуйте разрешить имя внутри Pod в кластере:

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

затем:

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

Вы увидите, что оно разрешается в `example.com`.

Аннотации для сервиса типа LoadBalancer

AWS EKS Cluster

Для подробного описания аннотаций сервиса LoadBalancer в EKS смотрите Annotation Usage Documentation ↗ .

Ключ	Значение	Описание
service.beta.kubernetes.io/aws-load-balancer-type	external: Использовать официальный AWS LoadBalancer Controller.	Определяет контроллер для типа LoadBalancer. **Примечание**: Пожалуйста, заранее свяжитесь с администратором платформы для развертывания AWS LoadBalancer Controller.
service.beta.kubernetes.io/aws-load-balancer-nlb-target-type	• instance: Трафик будет направляться на	Определяет, как трафик достигает Pod.

Ключ	Значение	Описание
	Pod через NodePort. • ip: Трафик направляется напрямую на Pod (кластер должен использовать Amazon VPC CNI).	
service.beta.kubernetes.io/aws-load-balancer-scheme	• internal: Частная сеть. • internet-facing: Публичная сеть.	Определяет, использовать ли частную или публичную сеть.
service.beta.kubernetes.io/aws-load-balancer-ip-address-type	• IPv4 • dualstack	Определяет поддерживаемый стек IP-адресов.

Huawei Cloud CCE Cluster

Для подробного описания аннотаций сервиса LoadBalancer в CCE смотрите Annotation Usage Documentation↗ .

Ключ	
kubernetes.io/elb.id	

Ключ	

kubernetes.io/elb.autocreate	Пример: `{"type":"public","bandwidth_name":"cce-bandwidt`
	`1551163379627","bandwidth_chargemode":"bandwidth","bandwic`
	`["cn-north-4b"],"l4_flavor_name":"L4_flavor.elb.s1.small"}`
	
	Примечание: Пожалуйста, сначала ознакомьтесь с ин
	параметры примера.

kubernetes.io/elb.subnet-id	
kubernetes.io/elb.class	• union: Общая балансировка нагрузки. • performance: Эксклюзивная балансировка нагрузки
kubernetes.io/elb.enterpriseID	

Ключ	

Azure AKS Cluster

Для подробного описания аннотаций сервиса LoadBalancer в AKS смотрите Annotation Usage Documentation ↗ .

Ключ	Значение	Описание
service.beta.kubernetes.io/azure-load-balancer-internal	• true: Частная сеть. • false: Публичная сеть.	Определяет, использовать ли частную или публичную сеть.

Google GKE Cluster

Для подробного описания аннотаций сервиса LoadBalancer в GKE смотрите Annotation Usage Documentation ↗ .

Ключ	Значение	Описание
networking.gke.io/load-balancer-type	Internal	Определяет использование частной сети.
loud.google.com/l4-rbs	enabled	По умолчанию публичный. Если этот параметр настроен, трафик будет направляться напрямую на Pod.

Создание Ingress

Правила Ingress (Kubernetes Ingress) открывают HTTP/HTTPS маршруты снаружи кластера для внутренней маршрутизации (Kubernetes Service), позволяя контролировать внешний доступ к вычислительным компонентам.

Создайте Ingress для управления внешним HTTP/HTTPS доступом к Service.

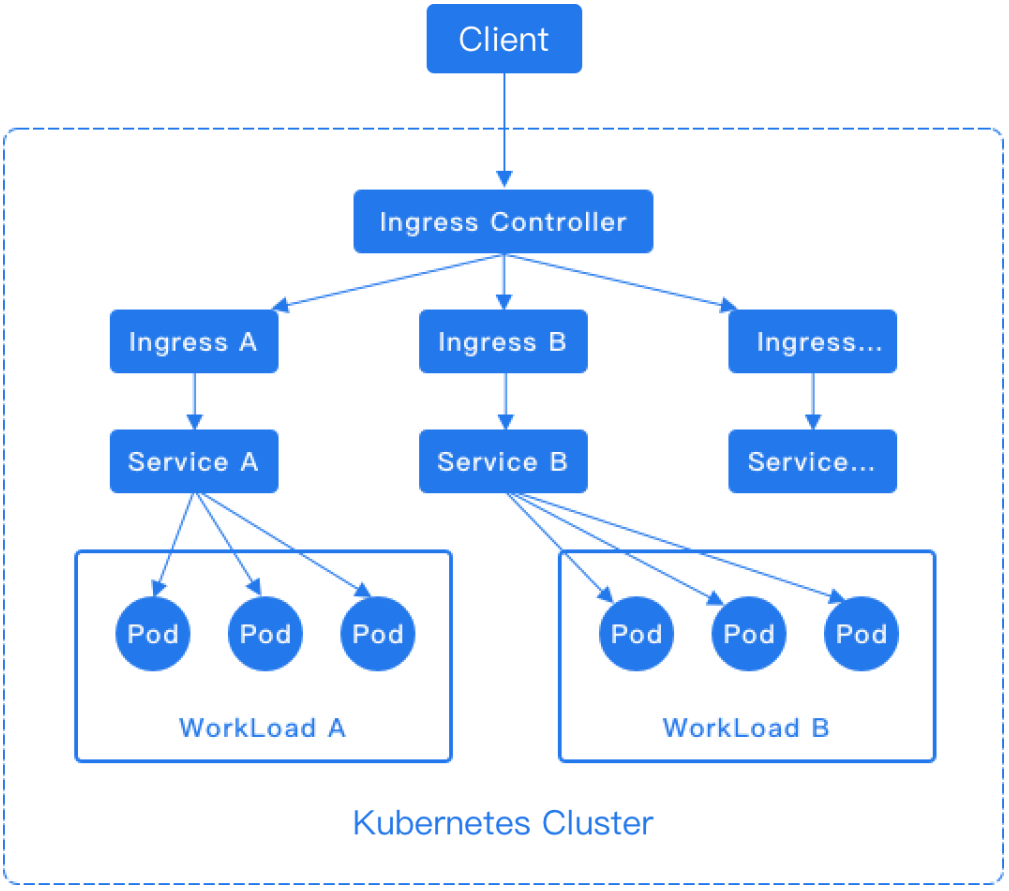
WARNING

При создании нескольких ingress в одном и том же namespace, разные ingress **НЕ ДОЛЖНЫ** иметь одинаковые **Domain**, **Protocol** и **Path** (то есть дублирование точек доступа не допускается).

Содержание

Метод реализации

Правила Ingress зависят от реализации Ingress Controller, который отвечает за отслеживание изменений в Ingress и Service. После создания нового Ingress, когда Ingress Controller получает запрос, он сопоставляет правило переадресации из Ingress и распределяет трафик по указанным внутренним маршрутам, как показано на схеме ниже.



NOTE

Для протокола HTTP Ingress поддерживает только порт 80 в качестве внешнего порта. Для протокола HTTPS Ingress поддерживает только порт 443 в качестве внешнего порта. Балансировщик нагрузки платформы автоматически добавляет порты 80 и 443 для прослушивания.

- установка ingress-nginx как ingress-controller
- установка alb как ingress-controller

Пример 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) Для получения дополнительной информации о конфигурациях см. nginx-configuration ↗.

(2) `nginx` используется для контроллера `ingress-nginx`, `$alb_name` — для использования alb в качестве ingress controller.

(3) Если вы хотите запускать ingress локально, предварительно настройте `hosts`.

Создание Ingress через веб-консоль

1. Зайдите в **Container Platform**.
2. В левой навигационной панели выберите **Network** > **Ingress**.
3. Нажмите **Create Ingress**.
4. Используйте инструкции ниже для настройки параметров.

Параметр	Описание
Ingress Class	Ingress может реализовываться разными контроллерами с разными именами `IngressClass`. Если на платформе доступно несколько ingress контроллеров, пользователь может выбрать нужный с помощью этой опции.
Domain Name	Хосты могут быть точными совпадениями (например, `foo.bar.com`) или с подстановочным знаком (например, `*.foo.com`). Доступные доменные имена выделяются администратором платформы.
Certificates	TLS секреты или сертификаты, выделенные администратором платформы.
Match Type и **Path**	• **Prefix**: Совпадение по префиксу пути, например, `/abcd` совпадает с `/abcd/efg` или `/abcde`. • **Exact**: Совпадение по точному пути, например, `/abcd`. • **Implementation specific**: Если вы используете кастомный Ingress controller для управления правилами Ingress, можно позволить контроллеру решать самостоятельно.
Service	Внешний трафик будет перенаправлен на этот Service.
Service Port	Укажите порт Service, на который будет перенаправлен трафик.

5. Нажмите **Create**.

Создание Ingress через CLI

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

Создание доменного имени

Добавьте ресурсы доменных имен на платформу и выделите домены для использования всеми проектами в кластере или ресурсами в конкретном проекте. При создании доменного имени поддерживается привязка сертификата.

NOTE

Созданные на платформе доменные имена должны быть разрешены на адрес балансировщика нагрузки кластера, прежде чем к ним можно будет получить доступ по доменному имени. Поэтому необходимо убедиться, что добавленные на платформу доменные имена успешно зарегистрированы и что доменные имена разрешаются на адрес балансировщика нагрузки кластера.

Успешно созданные и выделенные доменные имена на платформе могут использоваться в следующих функциях **Container Platform**:

- **Создание входящих правил**: **Network Management** > **Inbound Rules** > **Create Inbound Rule**
- **Создание нативных приложений**: **Application Management** > **Native Applications** > **Create Native Application** > **Add Inbound Rule**
- **Добавление прослушиваемых портов** для балансировки нагрузки: **Network Management** > **Load Balancer Details** > **Add Listening Port**

После привязки доменного имени к сертификату разработчики приложений могут просто выбрать доменное имя при настройке балансировщика нагрузки и входящих правил, что позволяет использовать сертификат, связанный с доменным именем, для поддержки https.

Содержание

Пример ресурса 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) Если включены сертификаты, необходимо заранее создать Secret типа LTS. `secret-ref` — это имя секрета.

Создание домена через веб-консоль

1. Перейдите в раздел **Administrator**.
2. В левой навигационной панели выберите **Network Management** > **Domain Names**.
3. Нажмите **Create Domain Name**.
4. Настройте соответствующие параметры согласно следующим инструкциям.

Параметр	Описание
Type	• Domain: Полное доменное имя, например, `developer.test.cn`. • Wildcard Domain: Подстановочный домен с символом подстановки (*), например, `*.test.cn`, который включает все поддомены домена `test.cn`.
Domain	Введите полное доменное имя или суффикс домена в зависимости от выбранного типа доменного имени.
Allocate Cluster	Если выделяется кластер, необходимо также выбрать проект, связанный с выделенным кластером, например, все проекты, связанные с кластером.
Certificate	Включает публичный ключ (tls.crt) и приватный ключ (tls.key) для создания сертификата, привязанного к доменному имени. Проект, которому выделен сертификат, совпадает с проектом привязанного доменного имени. **Примечания**: • Импорт бинарных файлов не поддерживается. • Привязанный сертификат должен соответствовать условиям корректного формата, быть в пределах срока действия и подписан для доменного имени и т. д. • После создания привязанного сертификата формат имени сертификата: доменное имя - случайные символы. • После создания привязанного сертификата он отображается в списке сертификатов, но обновление и удаление привязанного сертификата поддерживается только на странице деталей домена. • После создания привязанного сертификата поддерживается обновление содержимого сертификата, но замена другим сертификатом не поддерживается.

5. Нажмите **Create**.

Создание домена с помощью CLI

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

Последующие действия

- **Регистрация домена**: Зарегистрируйте домен, если созданный домен еще не зарегистрирован.
- **Разрешение домена**: Выполните разрешение домена, если домен не указывает на адрес балансировщика нагрузки кластера платформы.

Дополнительные ресурсы

- Configure Certificate

Создание сертификатов

После того как администратор платформы импортирует TLS-сертификат и назначит его определённому проекту, разработчики с соответствующими правами на проект смогут использовать сертификат, импортированный и назначенный администратором платформы, при работе с входящими правилами и функционалом балансировки нагрузки. В дальнейшем, например, при истечении срока действия сертификата, администратор платформы сможет централизованно обновить сертификат.

NOTE

Функциональность сертификатов в настоящее время не поддерживается для использования в кластерах публичного облака. При необходимости вы можете создавать секреты типа TLS в указанном namespace.

Содержание

Создание сертификата через веб-консоль

1. Перейдите в раздел **Administrator**.
2. В левой навигационной панели выберите **Network Management** > **Certificates**.
3. Нажмите **Create Certificate**.
4. Следуйте инструкциям ниже для настройки соответствующих параметров.

Параметр	Описание
Assign Project	• All Projects: Назначить сертификат для использования во всех проектах, связанных с текущим кластером. • Specified Project: Назначить сертификат для использования в указанном проекте. • No Assignment: Пока не назначать проект. После создания сертификата вы сможете обновить проекты, которые могут использовать сертификат, через операцию **Update Project**.
Public Key	Это tls.crt. При импорте открытого ключа бинарные файлы не поддерживаются.
Private Key	Это tls.key. При импорте закрытого ключа бинарные файлы не поддерживаются.

5. Нажмите **Create**.

Создание пула внешних IP-адресов

Пул внешних IP-адресов — это набор IP-адресов, которые MetalLB использует для получения внешних IP-адресов доступа для внутренних маршрутов типа LoadBalancer.

Содержание

Предварительные требования

Если необходимо использовать пул внешних IP-адресов типа BGP, обратитесь к администратору для включения соответствующих функций.

Ограничения и условия

IP-ресурсы для внешнего адреса должны соответствовать следующим условиям:

- Пул внешних адресов должен быть связан на уровне канального уровня (L2) с доступными узлами.
- IP-адреса должны быть пригодны для использования платформой и не могут включать IP-адреса, уже используемые физической сетью, например, IP-адреса шлюзов.
- Не должно быть пересечений с сетями, используемыми кластером, включая Cluster CIDR, Service CIDR, подсети и т. п.
- В среде с двойным стеком необходимо, чтобы в одном пуле внешних адресов одновременно присутствовали как IPv4, так и IPv6 адреса, и их количество было больше 0. В противном случае внутренние маршруты типа LoadBalancer с двойным стеком не смогут получить внешние адреса доступа.
- В среде IPv6 DNS узлов должен поддерживать IPv6, иначе плагин MetalLB не сможет быть успешно развернут.

Развертывание плагина MetalLB

Использование пула внешних адресов зависит от плагина MetalLB.

1. Перейдите в **Administrator**.
2. В левой навигационной панели выберите **Marketplace** > **Cluster Plugin**.
3. Найдите MetalLB, нажмите на **MetalLB** справа от ⋮ > **Deploy**.
4. Дождитесь, пока статус развертывания не изменится на **Deployment Successful**, чтобы завершить развертывание.

Пример ресурса custom resource (CR) IPAddressPool

```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
```

Режим BGP:

```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 Ссылка на пул IP-адресов.

INFO

Q: Что такое `L2Advertisement`?

A:

1. `L2Advertisement` — это Custom Resource (CRD), предоставляемый MetalLB для управления тем, какие адреса из пула IP-адресов должны транслироваться через ARP (IPv4) или NDP (IPv6) в режиме канального уровня (Layer 2).

Q: Какова цель `L2Advertisement`?

A:

1. Указать, какие IP-адреса из IPAddressPool должны транслироваться на уровне L2 (ARP/NDP объявления);
2. Контролировать поведение трансляции, чтобы предотвратить конфликты IP или трансляцию между сегментами;
3. Ограничивать область трансляции в средах с несколькими сетевыми интерфейсами и сетями.

Проще говоря, это говорит MetalLB, какие IP-адреса можно транслировать и кому (например, каким узлам).

Без определения `L2Advertisement` в режиме Layer2 MetalLB не будет транслировать никакие адреса.

Q: Что такое `BGPAdvertisement` в MetalLB?

A:

`BGPAdvertisement` — это Custom Resource Definition (CRD) Kubernetes, используемый в MetalLB↗, реализации балансировщика нагрузки для bare-metal Kubernetes кластеров. Он управляет тем, как диапазоны IP-адресов (определённые в `IPAddressPool`) анонсируются во внешние сети через BGP (Border Gateway Protocol).

Q: Почему `BGPAdvertisement` важен?

A:

В режиме BGP MetalLB контроллер устанавливает пиринг с внешними маршрутизаторами через BGP и анонсирует IP-адреса, назначенные объектам Kubernetes `Service`. Ресурс `BGPAdvertisement` позволяет:

- Управлять тем, какие пулы адресов анонсируются
- Настраивать параметры анонса маршрутов, такие как:
  - Агрегация маршрутов
  - BGP сообщества
  - Локальные предпочтения (приоритет BGP)

Без определения `BGPAdvertisement` MetalLB не будет анонсировать никакие адреса, даже если настроены BGP пиры.

Создание пула внешних IP-адресов через веб-консоль

1. Перейдите в **Administrator**.
2. В левой навигационной панели выберите **Network Management** > **External IP Address Pool**.
3. Нажмите **Create External IP Address Pool**.
4. Следуйте инструкциям для настройки параметров.

Параметр	Описание
Type	• L2: Коммуникация и пересылка на основе MAC-адресов, подходит для небольших или локальных сетей, требующих простой и быстрой коммутации на уровне 2, с преимуществами в простоте настройки и низкой задержке. • BGP (Alpha): Маршрутизация и пересылка на основе IP-адресов, использующая протокол BGP для обмена маршрутной информацией, подходит для крупных сетей с необходимостью сложной маршрутизации между несколькими автономными системами, с преимуществами в высокой масштабируемости и надежности.
IP Resources	Поддерживается ввод в форматах CIDR и диапазона IP. Нажмите **Add** для добавления нескольких записей, примеры: **CIDR**: `192.168.1.1/24`. **Диапазон IP**: `192.168.2.1` ~ `192.168.2.255`.
Available Nodes	В режиме L2 доступные узлы — это те, которые несут весь трафик VIP; в режиме BGP — узлы, которые несут VIP, устанавливают BGP-соединения с пирами и анонсируют маршруты внешне. • **Имя узла**: выбор доступных узлов по имени.

Параметр	Описание
	• **Выбор по меткам**: выбор доступных узлов по меткам. • **Показать детали узлов**: просмотр итогового списка доступных узлов. **Примечание**: • При использовании типа BGP доступные узлы — это узлы следующего хопа; убедитесь, что выбранные доступные узлы являются подмножеством BGP Connection Nodes. • Можно настроить либо выбор по меткам, либо по имени узла; если настроены оба варианта, итоговые доступные узлы — пересечение обоих.
BGP Peers	Выбор BGP пиров; подробности настройки см. в разделе BGP Peers.

5. Нажмите **Create**.

Создание пула внешних IP-адресов через CLI

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

Просмотр политики оповещений

1. Перейдите в **Administrator**.
2. В левой навигационной панели выберите **Network Management** > **External IP Address Pool**.
3. Нажмите **View Alarm Policy** в правом верхнем углу страницы, чтобы просмотреть общую политику оповещений для MetalLB.

Создание BGP-пиров

Узлы, которые устанавливают соединения для обмена маршрутной информацией либо между разными AS, либо внутри одного AS, используя протокол BGP.

Содержание

Терминология

Термин	Объяснение
AS Number	AS — это совокупность маршрутизаторов, управляемых одной технической административной организацией, использующей единую политику маршрутизации. Каждому AS в сети BGP присваивается уникальный номер AS для отличия от других AS. Номера AS делятся на 2-байтовые и 4-байтовые. • Диапазон 2-байтовых номеров AS — 1~65535, где 1~64511 — зарегистрированные публичные номера AS в Интернете, аналогичные публичным IP-адресам; 64512~65535 — приватные номера AS, аналогичные приватным IP-адресам. • Диапазон 4-байтовых номеров AS — 1~4294967295.

Термин	Объяснение
	Устройства, поддерживающие 4-байтовые номера AS, могут быть совместимы с устройствами, поддерживающими 2-байтовые номера AS.

Предварительные требования

Пожалуйста, свяжитесь с администратором для включения соответствующих функций.

Пример пользовательского ресурса BGPPeer (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"
```

Создание BGPPeer через веб-консоль

1. Перейдите в раздел **Administrator**.

2. В левой навигационной панели выберите **Network Management** > **BGP Peers**.

3. Нажмите **Create BGP Peer**.

4. Следуйте инструкциям ниже для настройки параметров.

Параметр	Описание
Local AS Number	Номер AS, к которому принадлежит узел, устанавливающий BGP-соединение. **Примечание**: Если нет особых требований, рекомендуется использовать конфигурацию IBGP, то есть локальный номер AS должен совпадать с номером AS пира.
Peer AS Number	Номер AS, к которому принадлежит BGP-пир.
Peer IP	IP-адрес BGP-пира, который должен быть действительным IP-адресом, способным установить BGP-соединение.
Local IP	IP-адрес узла, устанавливающего BGP-соединение. Если у узла несколько IP-адресов, выберите конкретный локальный IP для установления BGP-соединения с пиром.
Peer Port	Номер порта BGP-пира.
BGP Connected Node	Узел, который устанавливает BGP-соединение. Если этот параметр не задан, BGP-соединения будут устанавливаться со всеми узлами.
eBGP Multi-Hop	Позволяет устанавливать BGP-сессии между BGP-маршрутизаторами, которые не связаны напрямую. При включении этой функции значение TTL по умолчанию для BGP-пакетов равно 5, что позволяет устанавливать пиринговые отношения BGP через несколько промежуточных сетевых устройств, делая дизайн сети более гибким.
RouterID	32-битное числовое значение (обычно представляемое в формате с точками, аналогично формату IPv4-адреса), используемое для уникальной идентификации BGP-

Параметр	Описание
	маршрутизатора в сети BGP. Обычно применяется для установления соседских отношений BGP, обнаружения петель маршрутизации, выбора оптимальных путей и устранения неполадок в сети.

5. Нажмите **Create**.

Создание BGPPeer через CLI

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

Настройка подсетей

Содержание

Правила выделения IP

NOTE

Если проекту или пространству имён назначено несколько подсетей, IP-адрес будет случайным образом выбран из одной из подсетей.

- Выделение для проекта:
  - Если проект не привязан к подсети, Pods во всех пространствах имён этого проекта могут использовать IP-адреса только из подсети по умолчанию. Если в подсети по умолчанию недостаточно IP-адресов, Pods не смогут запуститься.
  - Если проект привязан к подсети, Pods во всех пространствах имён этого проекта могут использовать IP-адреса только из указанной подсети.
- Выделение для пространства имён:
  - Если пространство имён не привязано к подсети, Pods в этом пространстве имён могут использовать IP-адреса только из подсети по умолчанию. Если в подсети по умолчанию недостаточно IP-адресов, Pods не смогут запуститься.

- Если пространство имён привязано к подсети, Pods в этом пространстве имён могут использовать IP-адреса только из указанной подсети.

Сеть Calico

Создание подсетей в сети Calico для достижения более тонкой изоляции ресурсов внутри кластера.

Ограничения и особенности

В среде кластера с IPv6 подсети, созданные в сети Calico, по умолчанию используют инкапсуляцию VXLAN. Порты, необходимые для инкапсуляции VXLAN, отличаются от портов для инкапсуляции IPIP. Необходимо обеспечить открытие UDP порта 4789.

Пример custom resource (CR) подсети с сетью Calico

```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) Если `default` true, используется инкапсуляция VXLAN.

(2) См. параметры Encapsulation Mode и Encapsulation Protocol.

(3) См. параметры Outbound Traffic NAT.

Создание подсети в сети Calico через веб-консоль

1. Перейдите в **Administrator**.
2. В левой навигационной панели нажмите **Network Management** > **Subnets**.
3. Нажмите **Create Subnet**.
4. Настройте параметры согласно следующим инструкциям.

Параметр	Описание
CIDR	После назначения подсети проекту или пространству имён, контейнерные группы в этом пространстве будут случайным образом использовать IP-адреса из этого CIDR для связи. **Примечание**: Соответствие между CIDR и BlockSize см. в Reference Content.
Encapsulation Protocol	Выберите протокол инкапсуляции. **IPIP** не поддерживается в режиме dual-stack. • **IPIP**: реализует межсегментную связь с помощью протокола IPIP. • **VXLAN (Alpha)**: реализует межсегментную связь с помощью протокола VXLAN. • **No Encapsulation**: прямое соединение через маршрутизацию.
Encapsulation Mode	При выборе протокола инкапсуляции **IPIP** или **VXLAN** необходимо указать режим инкапсуляции, по умолчанию — **Always**. • **Always**: туннели IPIP / VXLAN всегда включены. • **Cross Subnet**: туннели IPIP / VXLAN включаются только при нахождении хостов в разных подсетях; при нахождении в одной подсети — прямое соединение через маршрутизацию.

Параметр	Описание
Outbound Traffic NAT	Выберите, включать ли NAT исходящего трафика (Network Address Translation), по умолчанию включено. Используется для установки адресов доступа, которые будут видны из внешней сети при выходе контейнерных групп подсети в интернет. При включённом NAT в качестве адреса доступа используется IP хоста; при отключённом NAT IP контейнерных групп подсети напрямую видны во внешней сети.

5. Нажмите **Confirm**.

6. На странице с деталями подсети выберите **Actions** > **Allocate Project** / **Allocate Namespace**.

7. Завершите настройку и нажмите **Allocate**.

Создание подсети в сети Calico через CLI

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

Reference Content

Динамическое соответствие между CIDR и blockSize представлено в таблице ниже.

CIDR	blockSize Size	Количество хостов	Размер одного пула IP
`prefix<=16`	26	1024+	64
`16<prefix<=19`	27	256~1024	32
`prefix=20`	28	256	16
`prefix=21`	29	256	8

CIDR	blockSize Size	Количество хостов	Размер одного пула IP
`prefix=22`	30	256	4
`prefix=23`	30	128	4
`prefix=24`	30	64	4
`prefix=25`	30	32	4
`prefix=26`	31	32	2
`prefix=27`	31	16	2
`prefix=28`	31	8	2
`prefix=29`	31	4	2
`prefix=30`	31	2	2
`prefix=31`	31	1	2

NOTE

Конфигурации подсетей с префиксом больше 31 не поддерживаются.

Сеть Kube-OVN

Создание подсети в оверлейной сети Kube-OVN для более тонкой изоляции ресурсов в кластере.

NOTE

Платформа имеет встроенную подсеть **join** для связи между узлами и Pods; избегайте конфликтов сетевых сегментов между **join** и вновь создаваемыми подсетями.

Пример custom resource (CR) подсети с оверлейной сетью Kube-OVN

```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) См. параметры Outbound Traffic NAT.

(2) См. параметры Reserved IP.

(3) См. параметры Gateway Type. Доступные значения: `distributed` или `centralized`.

(4) См. параметры Gateway Nodes.

(5) См. параметры ECMP. Требуется предварительное включение feature gate администратором.

Создание подсети в оверлейной сети Kube-OVN через веб-консоль

1. Перейдите в **Administrator**.
2. В левой навигационной панели нажмите **Network Management** > **Subnet**.
3. Нажмите **Create Subnet**.
4. Настройте параметры согласно следующим инструкциям.

Параметр	Описание
Network Segment	После назначения подсети проекту или пространству имён, IP-адреса из этого сегмента будут случайным образом выделяться для использования Pods.
Reserved IP	Указанные зарезервированные IP не будут автоматически выделяться. Например, могут использоваться как фиксированные IP для вычислительных компонентов.
Gateway Type	Выберите тип шлюза для подсети, управляющий исходящим трафиком. - **Distributed**: Каждый хост кластера может выступать в роли исходящего узла для Pods на текущем хосте, обеспечивая распределённый выход. - **Centralized**: Все Pods кластера используют один или несколько конкретных хостов как исходящие узлы, что упрощает аудит и контроль межсетевого экрана. Настройка нескольких централизованных **gateway nodes** обеспечивает высокую доступность.
ECMP (Alpha)	При выборе **Centralized** gateway можно использовать функцию ECMP. По умолчанию шлюз работает в режиме master-slave, только мастер-шлюз обрабатывает трафик. При включении ECMP (Equal-Cost Multipath Routing) исходящий трафик маршрутизируется по нескольким равнозначным путям ко всем доступным узлам шлюза, что увеличивает суммарную пропускную способность. **Примечание**: Необходимо предварительно включить соответствующие функции.
Gateway Nodes	При использовании **Centralized** gateway выберите один или несколько конкретных хостов в качестве узлов шлюза.
Outbound Traffic NAT	Выберите, включать ли NAT исходящего трафика (Network Address Translation). По умолчанию включено. Используется для установки адреса доступа, видимого из внешней сети при выходе Pods подсети в интернет.

Параметр	Описание
	При включённом NAT в качестве адреса доступа используется IP хоста; при отключённом NAT IP Pods подсети напрямую видны во внешней сети. В этом случае рекомендуется использовать централизованный шлюз.

5. Нажмите **Confirm**.
6. На странице с деталями подсети выберите **Actions** > **Allocate Project** / **Namespace**.
7. Завершите настройку и нажмите **Allocate**.

Создание подсети в оверлейной сети Kube-OVN через CLI

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

Underlay Network

Создание подсетей в Underlay-сети Kube-OVN не только обеспечивает более тонкую изоляцию ресурсов, но и улучшает производительность.

> **INFO**
>
> Контейнерная сеть в Kube-OVN Underlay требует поддержки физической сети. Рекомендуется ознакомиться с лучшими практиками Preparing the Kube-OVN Underlay Physical Network для обеспечения сетевой связности.

Инструкция по использованию

Общий процесс создания подсетей в сети Kube-OVN Underlay: Добавить Bridge Network > Добавить VLAN > Создать подсеть.

1. Имя сетевой карты по умолчанию.
2. Настройка сетевой карты по узлам.

Добавление Bridge Network через веб-консоль (опционально)

```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. Имя сетевой карты по умолчанию.
2. Настройка сетевой карты по узлам.

Bridge network — это мост, после привязки сетевой карты к мосту он может перенаправлять трафик контейнерной сети, обеспечивая связь с физической сетью.

Процедура:

1. Перейдите в **Administrator**.
2. В левой навигационной панели нажмите **Network Management** > **Bridge Network**.
3. Нажмите **Add Bridge Network**.
4. Настройте параметры согласно следующим инструкциям.

   Примечание:

   - *Target Pod* — все Pods, запланированные на текущем узле, или Pods в пространствах имён, привязанных к определённым подсетям, запланированным на текущем узле. Это зависит от области действия подсети под мостовой сетью.
   - Узлы в подсети Underlay должны иметь несколько сетевых карт, и сетевая карта, используемая мостовой сетью, должна быть выделена исключительно для

Underlay и не должна нести другой трафик, например SSH. Например, если в мостовой сети три узла, планирующие eth0, eth0, eth1 для исключительного использования Underlay, то сетевая карта по умолчанию может быть eth0, а для третьего узла — eth1.

Параметр	Описание
Default Network Card Name	По умолчанию целевой Pod будет использовать эту сетевую карту моста для связи с физической сетью.
Configure Network Card by Node	Целевые Pods на указанных узлах будут подключаться к указанной сетевой карте, а не к сетевой карте по умолчанию.
Exclude Nodes	Исключённые узлы не будут иметь мостового подключения для Pods. **Примечание**: Pods на исключённых узлах не смогут взаимодействовать с физической сетью или контейнерными сетями на других узлах, поэтому следует избегать планирования соответствующих Pods на эти узлы.

5. Нажмите **Add**.

Добавление Bridge Network через CLI

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

Добавление VLAN через веб-консоль (опционально)

```
# 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) Ссылка на мостовую сеть.

Платформа имеет преднастроенный виртуальный LAN **ovn-vlan**, который подключается к мостовой сети **provider**. Можно также создать новый VLAN, подключенный к другим мостовым сетям, обеспечивая изоляцию между VLAN.

Процедура:

1. Перейдите в **Administrator**.
2. В левой навигационной панели нажмите **Network Management** > **VLAN**.
3. Нажмите **Add VLAN**.
4. Настройте параметры согласно следующим инструкциям.

Параметр	Описание
VLAN ID	Уникальный идентификатор VLAN, используемый для различения виртуальных LAN.
Bridge Network	VLAN будет подключен к этой мостовой сети для связи с физической сетью.

5. Нажмите **Add**.

Добавление VLAN через CLI

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

Пример custom resource (CR) подсети с сетью Kube-OVN Underlay

```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) Ссылка на VLAN.

Создание подсети в сети Kube-OVN Underlay через веб-консоль

> **NOTE**
>
> Платформа также преднастроила подсеть **join** для связи между узлами и Pods в режиме Overlay. Эта подсеть не используется в режиме Underlay, поэтому важно избегать конфликтов IP-сегментов между **join** и другими подсетями.

Процедура:

1. Перейдите в **Administrator**.
2. В левой навигационной панели нажмите **Network Management** > **Subnet**.
3. Нажмите **Create Subnet**.

4. Настройте параметры согласно следующим инструкциям.

Параметр	Описание
VLAN	VLAN, к которому принадлежит подсеть.
Subnet	После назначения подсети проекту или пространству имён, IP-адреса из физической подсети будут случайным образом выделяться для Pods.
Gateway	Физический шлюз в указанной подсети.
Reserved IP	Указанные зарезервированные IP не будут автоматически назначаться. Например, могут использоваться как фиксированные IP для вычислительных компонентов.

5. Нажмите **Confirm**.

6. На странице с деталями подсети выберите **Action** > **Assign Project** / **Namespace**.

7. Завершите настройку и нажмите **Assign**.

Создание подсети в сети Kube-OVN Underlay через CLI

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

Связанные операции

При наличии в кластере подсетей Underlay и Overlay можно при необходимости настроить Автоматическую взаимосвязь между подсетями Underlay и Overlay.

Управление подсетями

Обновление шлюза через веб-консоль

Включает изменение метода исходящего трафика, узлов шлюза и конфигурации NAT.

1. Перейдите в **Administrator**.
2. В левой панели нажмите **Network Management** > **Subnets**.
3. Нажмите на имя подсети.
4. Выберите **Action** > **Update Gateway**.
5. Обновите параметры; подробности см. в Описание параметров.
6. Нажмите **OK**.

Обновление шлюза через 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}
]'
```

Обновление зарезервированных IP через веб-консоль

IP шлюза нельзя удалить из зарезервированных IP, остальные зарезервированные IP можно редактировать, удалять или добавлять.

1. Перейдите в **Administrator**.
2. В левой панели нажмите **Network Management** > **Subnets**.
3. Нажмите на имя подсети.
4. Выберите **Action** > **Update Reserved IP**.
5. После внесения изменений нажмите **Update**.

Обновление зарезервированных IP через 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"]
  }
]'
```

Назначение проектов через веб-консоль

Назначение подсетей конкретным проектам помогает командам лучше управлять и изолировать сетевой трафик для разных проектов, обеспечивая достаточные сетевые ресурсы для каждого проекта.

1. Перейдите в **Administrator**.
2. В левой панели нажмите **Network Management** > **Subnets**.
3. Нажмите на имя подсети.
4. Выберите **Action** > **Assign Project**.
5. После добавления или удаления проектов нажмите **Assign**.

Назначение проектов через CLI

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

Назначение пространств имён через веб-консоль

Назначение подсетей конкретным пространствам имён позволяет добиться более тонкой сетевой изоляции.

Примечание: Процесс назначения приведёт к перестройке шлюза, и исходящие пакеты будут отброшены! Убедитесь, что в данный момент нет бизнес-приложений, обращающихся к внешним кластерам.

1. Перейдите в **Administrator**.
2. В левой панели нажмите **Network Management** > **Subnets**.
3. Нажмите на имя подсети.
4. Выберите **Action** > **Assign Namespace**.
5. После добавления или удаления пространств имён нажмите **Assign**.

Назначение пространств имён через CLI

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

Расширение подсетей через веб-консоль

Когда диапазон зарезервированных IP подсети достигает предела или близок к исчерпанию, его можно расширить на основе исходного диапазона подсети без влияния на нормальную работу существующих сервисов.

1. Перейдите в **Administrator**.
2. В левой панели нажмите **Network Management** > **Subnets**.
3. Нажмите на имя подсети.

4. Выберите **Action** > **Expand Subnet**.

5. Завершите настройку и нажмите **Update**.

Расширение подсетей через CLI

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

Управление сетями Calico

Поддерживается назначение проектов и пространств имён; подробности см. в разделах назначение проектов и назначение пространств имён.

Удаление подсети через веб-консоль

NOTE

- При удалении подсети, если есть контейнерные группы, использующие IP из этой подсети, они продолжат работать с теми же IP, но не смогут обмениваться трафиком по сети. Контейнерные группы можно пересоздать для использования IP из подсети по умолчанию или назначить новое пространство подсети для пространства имён, в котором они находятся.
- Подсеть по умолчанию удалить нельзя.

1. Перейдите в **Administrator**.

2. В левой навигационной панели нажмите **Network Management** > **Subnets**.

3. Нажмите ⋮ > **Delete** и подтвердите удаление.

Удаление подсети через CLI

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

Создание сетевых политик

INFO

Платформа теперь предоставляет два разных интерфейса для работы с сетевыми политиками. Старый интерфейс поддерживается для совместимости, а новый более гибкий и включает встроенный редактор YAML. Рекомендуется использовать новую версию.

Пожалуйста, свяжитесь с администратором платформы, чтобы включить feature gate `network-policy-next` для доступа к новому интерфейсу.

NetworkPolicy — это ресурс Kubernetes с областью действия в пределах namespace, реализуемый плагинами CNI. С помощью сетевых политик можно контролировать сетевой трафик Pod-ов, обеспечивая сетевую изоляцию и снижая риск атак.

По умолчанию все Pod-ы могут свободно общаться, разрешая входящий и исходящий трафик из любых источников. При применении NetworkPolicy целевые Pod-ы будут принимать только трафик, соответствующий спецификации.

WARNING

Сетевые политики применяются только к трафику контейнеров. Они не влияют на Pod-ы, работающие в режиме **hostNetwork**.

Пример 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` и `to` поддерживают `namespaceSelector`, `podSelector`, `ipBlock`

Содержание

Создание NetworkPolicy через веб-консоль

1. Войдите в **Container Platform**.

2. В левой навигационной панели выберите **Network** > **Network Policies**.

3. Нажмите **Create Network Policy**.

4. Следуйте инструкциям ниже для заполнения соответствующих настроек.

Область	Параметр	Описание
Целевой Pod	Селектор Pod	Введите метки целевых Pod-ов в формате ключ-значение; если не задано, политика применяется ко всем Pod-ам в текущем namespace.
	Предварительный просмотр целевых Pod-ов, на которые влияет текущая политика	Нажмите **Preview**, чтобы увидеть целевые Pod-ы, на которые распространяется эта сетевая политика.
Ingress	Блокировать весь входящий трафик	Блокирует весь входящий трафик к целевому Pod-у. **Примечание**: • Если в поле `spec.policyTypes` в YAML добавлен Ingress без настройки конкретных правил, при возврате к форме опция **Блокировать весь входящий трафик** будет автоматически выбрана.

Область	Параметр		Описание
			• Если поля `spec.ingress`, `spec.egress` и `spec.policyTypes` одновременно удалены в YAML, при возврате к форме опция **Блокировать весь входящий трафик** будет автоматически выбрана.
	Правила **Описание**: Если в правилах добавлено несколько источников, между ними действует логическая операция **ИЛИ**.	Pod-ы в текущем namespace	Соответствуют Pod-ам с указанными метками в текущем namespace; только такие Pod-ы могут обращаться к целевому Pod-у. Можно нажать **Preview**, чтобы увидеть Pod-ы, на которые влияет текущее правило. Если этот параметр не настроен, по умолчанию все Pod-ы в текущем namespace имеют доступ к целевому Pod-у.
		Pod-ы в текущем кластере	Соответствуют namespace или Pod-ам с указанными метками в кластере; только такие Pod-ы могут обращаться к целевому Pod-у. Можно нажать **Preview**, чтобы увидеть Pod-ы, на которые влияет текущее правило. • Если настроены и селектор namespace, и селектор Pod, будет взято пересечение — то есть выбраны Pod-ы с указанными метками из указанного namespace. • Если этот параметр не настроен, по умолчанию все

Область	Параметр		Описание
			Pod-ы из всех namespace в кластере имеют доступ к целевому Pod-у.
		IP-диапазон	Введите CIDR, который может обращаться к целевому Pod-у, и можно исключить CIDR-диапазоны, которым доступ запрещён. Если этот параметр не настроен, любой трафик может обращаться к целевому Pod-у. **Описание**: Можно добавить исключения в формате *example_ip*/32 для исключения отдельного IP-адреса.
	Порт		Соответствует трафику на указанных протоколах и портах; можно добавить числовые порты или имена портов Pod-ов. Если этот параметр не настроен, будут соответствовать все порты.

Область	Параметр	Описание
Egress	Блокировать весь исходящий трафик	Блокирует весь исходящий трафик от целевого Pod-а. **Примечание**: • Если в поле `spec.policyTypes` в YAML добавлен Egress без настройки конкретных правил, при возврате к форме опция **Блокировать весь исходящий трафик** будет автоматически выбрана.
	Другие параметры	Аналогично параметрам **Ingress**, здесь не будет подробно описано.

1. Нажмите **Create**.

Создание NetworkPolicy с помощью CLI

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

Справка

Для получения дополнительной информации ознакомьтесь с официальной документацией по Network Policies↗.

Создание Admin Network Policies

INFO

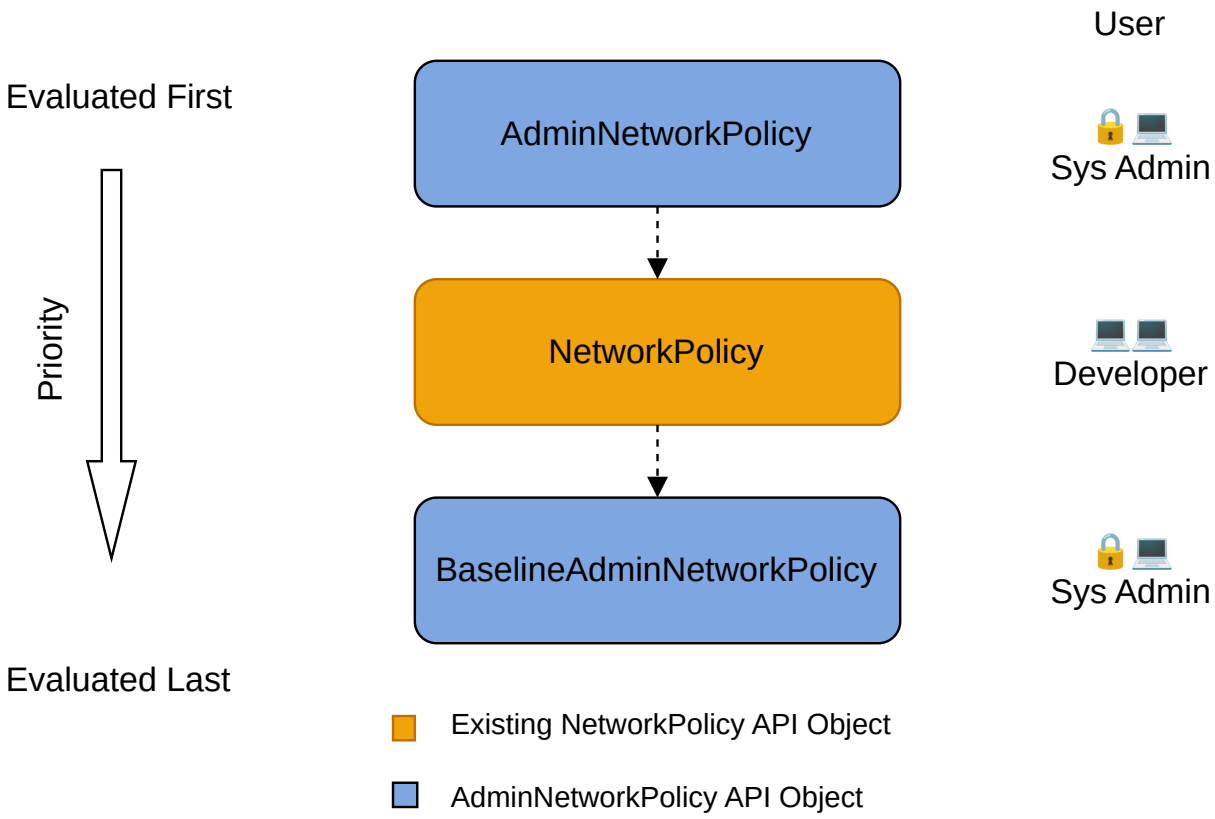
Платформа теперь предоставляет два разных интерфейса для Cluster Network Policies. Старый интерфейс поддерживается для совместимости, в то время как новый более гибкий и предоставляет нативный YAML-редактор. Рекомендуется использовать новую версию.

Пожалуйста, свяжитесь с администратором платформы для включения feature-gate `cluster-network-policy` и `cluster-network-policy-next` для доступа к новому интерфейсу.

Новая cluster network policy использует стандартный дизайн Kubernetes сообщества Admin Network Policy↗, предоставляя более гибкие методы конфигурации и богатые опции настройки.

При применении нескольких сетевых политик соблюдается строгий порядок приоритетов: Admin Network Policy имеет приоритет над Network Policy, которая, в свою очередь, имеет приоритет над Baseline Admin Network Policy.

Процедура следующая：



Содержание

Примечания

- Административные сетевые политики поддерживаются только Kube-OVN CNI.
- В сетевом режиме Kube-OVN эта функция находится на уровне Alpha.
- В кластере может существовать только одна Baseline Admin Network Policy.

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) Чем меньше число, тем выше приоритет.

2 `subject`: можно указать не более одного селектора — либо namespaceSelector, либо podSelector.

3 `action`: допустимые значения — Allow, Deny и Pass. Allow разрешает доступ трафика, Deny запрещает доступ, Pass разрешает трафик и пропускает последующие политики с более низким приоритетом, позволяя обработку трафика другими политиками (NetworkPolicy и BaselineAdminNetworkPolicy).

4 Допустимые значения — Namespace Selector, Pod Selector.

5 Допустимые значения — 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) В кластере может быть создана только одна baseline admin network policy с metadata.name= `default`.

(2) Допустимые значения — Allow, Deny.

Создание AdminNetworkPolicy или BaselineAdminNetworkPolicy через веб-консоль

1. Перейдите в раздел **Administrator**.
2. В левой навигационной панели выберите **Network** > **Cluster Network Policies**.
3. Нажмите **Create Admin Network Policies** или **Configure the Baseline Admin Network Policy**.
4. Следуйте инструкциям ниже для завершения соответствующей настройки.

Область	Параметр	Описание
Основная информация	Имя	Имя Admin Network Policy или Baseline Admin Network Policy.
	Приоритет	Определяет порядок оценки и применения политик. Меньшие числовые значения означают более высокий приоритет. **Примечание:** baseline admin network policy не имеет приоритета.
Целевой Pod	Namespace Selector	Введите метки целевых Namespace в формате ключ-значение. Если не указано, политика применяется ко всем Namespace в текущем кластере. При указании политика применяется только к Pod в Namespace, соответствующих этим селекторам.

Область	Параметр	Описание
	Просмотр целевых Pod, затронутых текущей политикой	Нажмите **Preview**, чтобы увидеть целевые Pod, на которые влияет эта сетевая политика.
	Pod Selector	Введите метки целевых Pod в формате ключ-значение. Если не указано, политика применяется ко всем Pod в текущем namespace.
	Просмотр целевых Pod, затронутых текущей политикой	Нажмите **Preview**, чтобы увидеть целевые Pod, на которые влияет эта сетевая политика.
Ingress	Действие с трафиком	Определяет, как обрабатывать входящий трафик к целевым Pod. Имеет три режима: **Allow** (разрешить трафик), **Deny** (запретить трафик) и **Pass** (пропустить все административные политики с более низким приоритетом, позволяя обработку трафика Network Policy или, если Network Policy отсутствует, Baseline Admin Network Policy). **Примечание:** baseline admin network policy не поддерживает действие **Pass**.

Область	Параметр		Описание
	Правило **Описание**: Если в правиле добавлено несколько источников, между ними действует логическая операция **ИЛИ**.	Pod Selector	Соответствует namespace или Pod с указанными метками в кластере; доступ к целевому Pod разрешён только для соответствующих Pod. Можно нажать **Preview**, чтобы увидеть Pod, затронутые текущим правилом. • Если настроены и namespace, и Pod селекторы, берётся их пересечение, то есть выбираются Pod с указанными метками из указанных namespace. • Если этот параметр не настроен, по умолчанию все Pod во всех namespace кластера могут получить доступ к целевому Pod.
		Namespace Selector	Соответствует Pod с указанными метками в текущем namespace; доступ к целевому Pod разрешён только для соответствующих Pod. Можно нажать **Preview**, чтобы увидеть Pod, затронутые текущим правилом. Если этот параметр не настроен, по

Область	Параметр		Описание
			умолчанию все Pod в текущем namespace могут получить доступ к целевому Pod.
	Порты		Соответствует трафику на указанных протоколах и портах; можно добавлять числовые порты или имена портов на Pod. Если этот параметр не настроен, будут соответствовать все порты.
Egress	Правило **Описание**: Если в правиле добавлено несколько источников, между ними действует логическая операция **ИЛИ**.	Node Selector	Определяет, к каким IP-адресам узлов целевые Pod имеют доступ. Можно выбрать узлы по их меткам, чтобы контролировать доступные IP-адреса узлов для Pod.
		IP Range	Укажите CIDR-диапазоны, к которым целевые Pod разрешено подключаться. Если этот параметр не настроен, по умолчанию целевые Pod могут подключаться к любому IP.
		Другие параметры	Аналогично параметрам Ingress, с теми же опциями настройки и поведением.

Создание AdminNetworkPolicy или BaselineAdminNetworkPolicy через CLI

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

Дополнительные ресурсы

- Configure Cluster Network Policies

Настройка сети Kube-OVN для поддержки нескольких сетевых интерфейсов Pod (Alpha)

Используя Multus CNI, вы можете добавить несколько сетевых интерфейсов с разными сетями к Pod. Используйте CRD Subnet и IP сети Kube-OVN для расширенного управления IP, реализуя управление подсетями, резервирование IP, случайное распределение, фиксированное распределение и другие функции.

Содержание

Установка Multus CNI

Развертывание плагина Multus CNI

1. Перейдите в раздел **Administrator**.

2. В левой навигационной панели нажмите **Marketplace** > **Cluster Plugins**.

3. В строке поиска введите "multus", чтобы найти плагин Multus CNI.
4. Найдите в списке плагин **"Alauda Container Platform Networking for Multus"**.
5. Нажмите на три точки (⋮) рядом с записью плагина и выберите **Install**.
6. Плагин будет развернут в вашем кластере. Вы можете отслеживать статус установки в колонке **State**.

> **NOTE**
>
> Плагин Multus CNI служит промежуточным звеном между другими CNI плагинами и Kubernetes, позволяя Pod иметь несколько сетевых интерфейсов.

Создание подсетей

Создайте подсеть attachnet согласно следующему примеру: `network-attachment-definition.yml`.

> **NOTE**
>
> Формат провайдера в config — `<NAME>.<NAMESPACE>.ovn`, где `<NAME>` и `<NAMESPACE>` — имя и namespace данного CR NetworkAttachmentDefinition соответственно.

```
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"
    }'
```

После создания примените ресурс:

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

Используйте следующий пример для создания подсети Kube-OVN для второго сетевого интерфейса: `subnet.yml`.

NOTE

- `spec.provider` должен совпадать с провайдером в NetworkAttachmentDefinition.
- Если необходимо использовать Underlay подсеть, установите `spec.vlan` подсети в имя CR VLAN, который вы хотите использовать. Настройте остальные параметры подсети по необходимости.

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

После создания примените ресурс:

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

Создание Pod с несколькими сетевыми интерфейсами

Создайте Pod согласно следующему примеру.

NOTE

- В `metadata.annotations` должен содержаться ключ-значение `k8s.v1.cni.cncf.io/networks=default/attachnet`, где формат значения — `<NAMESPACE>/<NAME>`, а `<NAMESPACE>` и `<NAME>` — namespace и имя CR NetworkAttachmentDefinition соответственно.
- Если Pod требует три сетевых интерфейса, настройте значение `k8s.v1.cni.cncf.io/networks` как `default/attachnet,default/attachnet2`.

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

После успешного создания Pod используйте команду `kubectl exec pod1 -- ip a`, чтобы просмотреть IP-адреса Pod.

Проверка создания двух сетевых интерфейсов

Используйте следующую команду, чтобы проверить успешное создание двух сетевых интерфейсов:

```
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
```

Дополнительные возможности

Фиксированный IP

- **Основной сетевой интерфейс (первый интерфейс)**: Если необходимо зафиксировать IP основного сетевого интерфейса, метод такой же, как и при

использовании фиксированного IP с одним сетевым интерфейсом. Добавьте аннотацию `ovn.kubernetes.io/ip_address=<IP>` к Pod.

- **Вторичный сетевой интерфейс (второй или другие интерфейсы)**: Основной метод аналогичен основному сетевому интерфейсу, с тем отличием, что `ovn` в ключе аннотации заменяется на провайдера соответствующего NetworkAttachmentDefinition. Пример: `attachnet.default.ovn.kubernetes.io/ip_address=172.170.0.101`.

Дополнительные маршруты

Начиная с версии 1.8.0, Kube-OVN поддерживает настройку дополнительных маршрутов для вторичных сетевых интерфейсов. При использовании этой функции добавьте поле `routes` в config NetworkAttachmentDefinition и заполните маршруты, которые необходимо настроить. Пример:

```
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"
        }
      ]
    }'
```

Настройка сетевых политик кластера

Сетевые политики кластера отвечают за управление правилами контроля доступа **на уровне проектов**. При включении этой функции разные проекты по умолчанию изолированы друг от друга, и вычислительные компоненты в разных проектах не могут взаимодействовать по сети. Связь может быть организована путем добавления правил **Доступ между проектами** или **Доступ по IP-адресам**.

После настройки сетевые политики кластера будут синхронизированы с namespace-ами в кластере и могут быть просмотрены в модуле функции **Network Policies** на платформе контейнеров.

Содержание

Примечания

- Эффективность сетевых политик кластера зависит от того, поддерживает ли используемый в кластере сетевой плагин сетевые политики.
  - Kube-OVN и Calico поддерживают сетевые политики.
  - Flannel не поддерживает сетевые политики.
  - При доступе к кластеру или использовании кастомного сетевого плагина можно обратиться к соответствующей документации для подтверждения поддержки.
- Функциональность находится на стадии Alpha при использовании сетевого режима Kube-OVN.

Процедура

1. Перейдите в раздел **Administrator**.

2. В левой навигационной панели выберите **Networking** > **Cluster Network Policies**.

3. Нажмите **Configure Now**.

4. Следуйте инструкциям ниже для завершения соответствующей настройки.

Параметр настройки	**Описание**
Полная изоляция между проектами	Включение или отключение переключателя полной изоляции между проектами, который включен по умолчанию и может быть выключен нажатием. При включении достигается сетевая изоляция между всеми проектами текущего кластера, и другим ресурсам запрещен доступ к любому проекту внутри кластера (например, внешним IP, балансировщикам нагрузки). Это не влияет на доступ проектов к ресурсам вне кластера.
Доступ одного проекта	Этот параметр действует только при включенном переключателе **Полная изоляция между проектами**. Настройте **исходный проект** и **целевой проект** для одностороннего доступа. Нажмите **Add** для добавления записи конфигурации, поддерживается несколько записей. В выпадающем списке **исходный проект** выберите проект, который будет иметь доступ к целевому проекту, или выберите все проекты; в выпадающем списке **целевой проект** выберите проект, к которому будет осуществлен доступ.
Доступ по IP-сегменту	Этот параметр действует только при включенном переключателе **Полная изоляция между проектами**. Настройте конкретный **IP/сегмент** и **целевой проект** для одностороннего доступа. Нажмите **Add** для добавления записи конфигурации, поддерживается несколько записей.

Параметр настройки	Описание
	В поле ввода **исходный IP-сегмент** введите IP-адрес или CIDR-сегмент, который будет иметь доступ к целевому проекту; в выпадающем списке **целевой проект** выберите проект, к которому будет осуществлен доступ.

5. Нажмите **Configure**.

Настройка Egress Gateway

Содержание

Описание Egress Gateway

Egress Gateway используется для контроля доступа Pods к внешней сети с помощью группы статических адресов и обладает следующими особенностями:

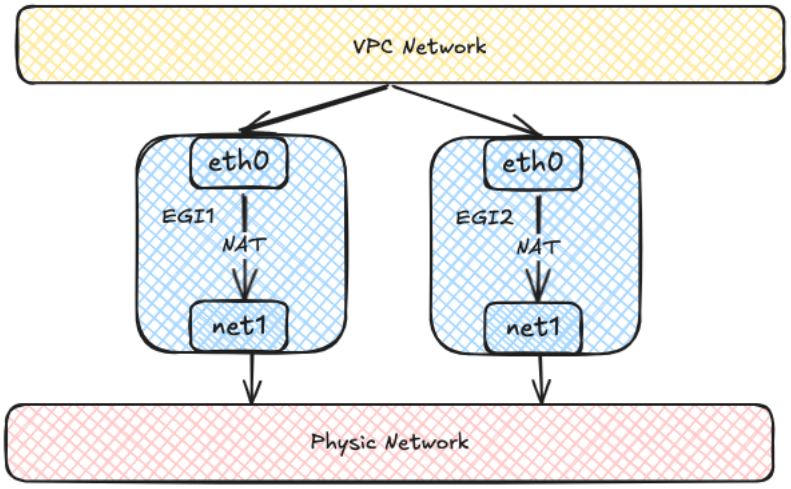
- Обеспечивает высокую доступность Active-Active через ECMP, позволяя горизонтально масштабировать пропускную способность
- Реализует быстрое переключение при сбое (<1 с) с помощью BFD
- Поддерживает IPv6 и dual-stack
- Позволяет тонко управлять маршрутизацией через NamespaceSelector и PodSelector
- Обеспечивает гибкое планирование Egress Gateway с помощью NodeSelector

В то же время Egress Gateway имеет следующие ограничения:

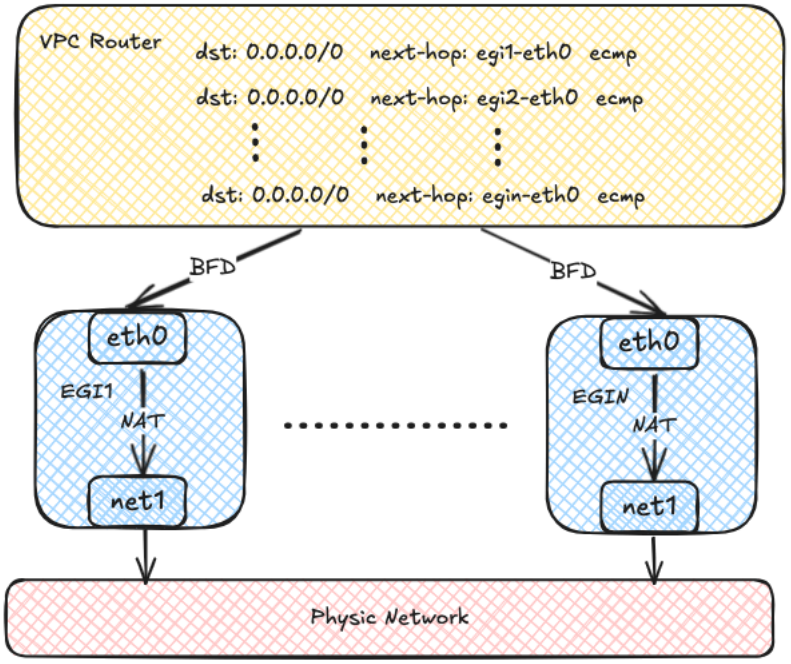
- Использует macvlan для базового сетевого подключения, требуя поддержки Underlay со стороны физической сети
- В режиме многократных экземпляров Gateway требуется несколько Egress IP
- В настоящее время поддерживается только SNAT; EIP и DNAT не поддерживаются
- В настоящее время не поддерживается запись отношений трансляции исходных адресов

Детали реализации

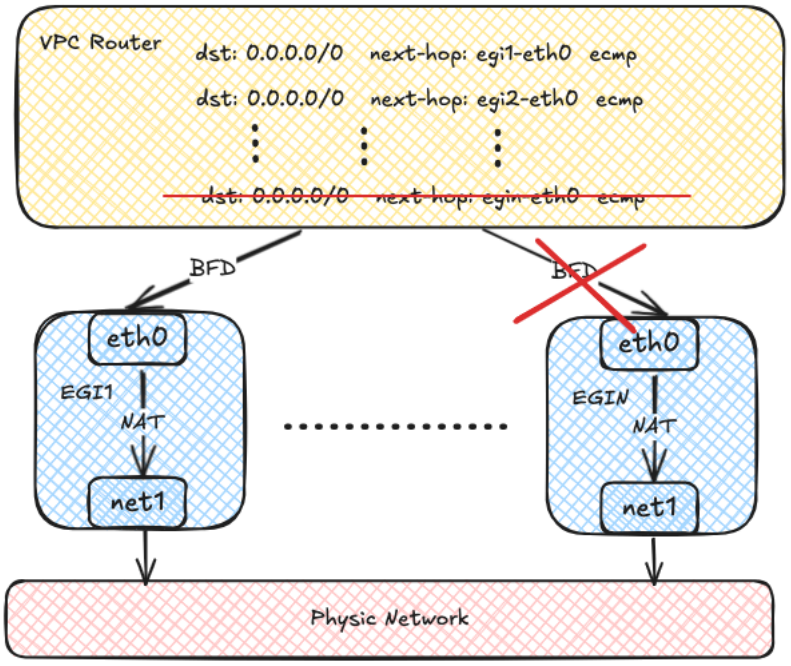
Каждый Egress Gateway состоит из нескольких Pods с несколькими сетевыми интерфейсами. Каждый Pod имеет два сетевых интерфейса: один подключается к виртуальной сети для связи внутри VPC, а другой соединяется с физической сетью через Macvlan для внешней сетевой коммуникации. Трафик виртуальной сети в конечном итоге выходит во внешнюю сеть через NAT внутри экземпляров Egress Gateway.



Каждый экземпляр Egress Gateway регистрирует свой адрес в таблице маршрутизации OVN. Когда Pod внутри VPC должен получить доступ к внешней сети, OVN использует хеширование исходного адреса для передачи трафика на несколько адресов экземпляров Egress Gateway, обеспечивая балансировку нагрузки. С увеличением количества экземпляров Egress Gateway пропускная способность также масштабируется горизонтально.



OVN использует протокол BFD для опроса нескольких экземпляров Egress Gateway. При сбое экземпляра Egress Gateway OVN помечает соответствующий маршрут как недоступный, обеспечивая быстрое обнаружение и восстановление после отказа.



Примечания

- Поддержка Egress Gateway есть только в Kube-OVN CNI.
- Для работы Egress Gateway требуется Multus-CNI.

Использование

Создание Network Attachment Definition

Egress Gateway использует несколько сетевых интерфейсов для доступа как к внутренней, так и к внешней сети, поэтому необходимо создать Network Attachment Definition для подключения к внешней сети. Пример использования плагина macvlan с IPAM от Kube-OVN приведён ниже:

```
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) Интерфейс хоста, который подключается к внешней сети.

(2) Имя провайдера в формате `<network attachment definition name>.<namespace>`.

(3) Имя провайдера, используемое для идентификации внешней сети и ДОЛЖНО совпадать с указанным в NetworkAttachmentDefinition.

TIP

Вы можете создать Network Attachment Definition с любым CNI плагином для доступа к соответствующей сети.

Создание VPC Egress Gateway

Создайте ресурс VPC Egress Gateway, как показано в примере ниже:

```yaml
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, в котором создаются экземпляры VPC Egress Gateway.
2. Количество реплик экземпляров VPC Egress Gateway.
3. Внешний подсеть, подключаемая к внешней сети.
4. Node selectors, к которым применяется VPC Egress Gateway.
5. Namespace и Pod селекторы, к которым применяется VPC Egress Gateway.
6. Политики для VPC Egress Gateway, включая SNAT и применяемые подсети/ipBlocks.
7. Включение SNAT для политики.
8. Подсети, к которым применяется политика.
9. IP-блоки, к которым применяется политика.

Данный ресурс создаёт VPC Egress Gateway с именем *gateway1* в namespace default, и следующие Pods будут выходить во внешнюю сеть через подсеть *macvlan1*:

- Pods в namespace default.
- Pods в подсети *subnet1*.
- Pods с IP в CIDR *10.18.0.0/16*.

> **NOTICE**
>
> Pods, соответствующие *.spec.selectors*, всегда будут выходить во внешнюю сеть с включённым SNAT.

После создания проверьте ресурс VPC Egress Gateway:

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

Для просмотра дополнительной информации:

```
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
```

Для просмотра 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>
```

Для просмотра IP-адресов, маршрутов и правил iptables в 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
```

Захват пакетов в Pod Gateway для проверки сетевого трафика:

```
$ 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
```

Политики маршрутизации автоматически создаются на 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) Политика Logical Router, используемая VPC Egress Gateway для перенаправления трафика от Pods, указанных в *.spec.policies*.

(2) Политика Logical Router, используемая VPC Egress Gateway для перенаправления трафика от Pods, указанных в *.spec.selectors*.

Если необходимо включить балансировку нагрузки, измените *.spec.replicas*, как показано в примере:

```
$ 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
```

Включение высокой доступности на основе BFD

Высокая доступность на основе BFD опирается на функцию VPC BFD LRP, поэтому необходимо изменить ресурс VPC для включения BFD Port. Пример включения BFD Port для 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) Включение BFD Port.

(2) IP-адрес BFD Port, который ДОЛЖЕН быть валидным и не конфликтовать с другими IP/подсетями.

(3) Node selector для выбора узлов, на которых BFD Port работает в режиме Active-Backup.

После включения BFD Port на соответствующем OVN Logical Router автоматически создаётся LRP, выделенный для BFD:

```
$ 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, созданный на OVN Logical Router.

Далее установите *.spec.bfd.enabled* в *true* в VPC Egress Gateway. Пример:

```
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, к которому принадлежит Egress Gateway.

2 Внутренняя подсеть, к которой подключены экземпляры Egress Gateway.

3 Внешняя подсеть, к которой подключены экземпляры Egress Gateway.

4 Включение BFD для Egress Gateway.

5 Минимальный интервал приёма BFD в миллисекундах.

6 Минимальный интервал передачи BFD в миллисекундах.

7 Множитель BFD, определяющий количество пропущенных пакетов до фиксации сбоя.

Для просмотра информации о VPC Egress Gateway:

```
$ 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
```

Для просмотра BFD-сессий:

```
$ 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

Если все экземпляры шлюза недоступны, исходящий трафик, к которому применяется VPC Egress Gateway, будет отброшен.

Параметры конфигурации

VPC BFD Port

Поля		Тип	Необязательно	Значение по умолчанию	Описание		
enabled		boolean	Да	false	Включение BFD Port.		true
ip		string	Нет	-	IP-адрес, используемый BFD Port. Не должен конфликтовать с другими адресами. Поддерживаются IPv4, IPv6 и dual-stack.		169 fdff 169
nodeSelector	matchLabels	object	Да	-	Селекторы меток для выбора узлов, на которых работает BFD Port. BFD Port привязывается к OVN HA Chassis Group выбранных узлов и работает в режиме Active/Backup. Если поле не указано, Kube-OVN автоматически выбирает до трёх узлов. Все ресурсы OVN HA Chassis Group можно просмотреть командой *kubectl ko nbctl list ha_chassis_group*.	Карта пар {ключ,значение}.	-
	matchExpressions	object array	Да	-		Список требований селектора меток. Требования объединяются логическим И.	-

VPC Egress Gateway

Поля	Тип	Необязательно	Значение по умолчанию	
vpc	string	Да	Имя default VPC (ovn-	Имя VPC.

Поля			Тип	Необязательно	Значение по умолчанию	
					cluster)	
replicas			integer/int32	Да	1	Количество реплик
prefix			string	Да	-	Неизменяемый пре
image			string	Да	-	Образ, используем
internalSubnet			string	Да	Имя default подсети внутри VPC.	Имя подсети для д
externalSubnet				Нет	-	
internalIPs			string array	Да	-	IP-адреса для дост Поддерживаются IF Количество указани Рекомендуется ука крайних случаев с
externalIPs						
bfd	enabled		boolean	Да	false	Конфигурация BFD
	minRX		integer/int32	Да	1000	
	minTX					
	multiplier		integer/int32	Да	3	
policies	snat		boolean	Да	false	Политики исходящего трафика.
	ipBlocks		string array	Да	-	
	subnets		string array	Да	-	
selectors	namespaceSelector	matchLabels	object	Да	-	Настройка политик исходящего трафи по селекторам namespace и Pod. SNAT/MASQUERAI применяется к совпадающим Pods
		matchExpressions	object array	Да	-	

Поля			Тип	Необязательно	Значение по умолчанию	
	podSelector	matchLabels	object	Да	-	
		matchExpressions	object array	Да	-	
nodeSelector	matchLabels		object	Да	-	Node selector для выбора узлов, на которых развертывается workload. Workload (Deployment/Pod) будет запускаться н выбранных узлах.
	matchExpressions		object array	Да	-	
	matchFields		object array	Да	-	
trafficPolicy			string	Да	Cluster	**Эффективно толь** Доступные значени При установке в *Lo* перенаправляться работающий на том Если экземпляр не на другие экземпля

Дополнительные ресурсы

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

Наблюдаемость сети

Содержание

О DeepFlow

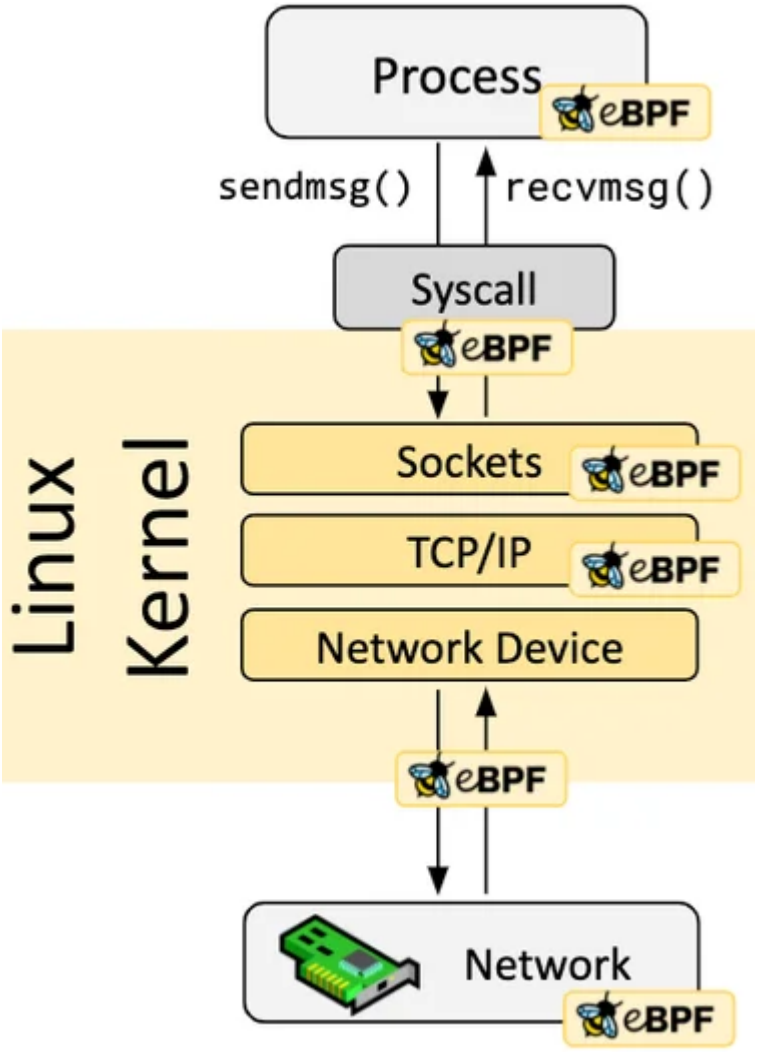
Что такое DeepFlow

Проект с открытым исходным кодом DeepFlow направлен на обеспечение глубокой наблюдаемости сложных облачных и AI-приложений. DeepFlow реализует сбор данных

без написания кода с помощью eBPF для метрик, распределённого трассирования, логов запросов и профилирования функций, а также интегрируется с SmartEncoding для достижения полной корреляции Full Stack и эффективного доступа ко всем данным наблюдаемости. С DeepFlow облачные и AI-приложения автоматически получают глубокую наблюдаемость, исключая необходимость постоянного инструментирования кода разработчиками и предоставляя возможности мониторинга и диагностики, охватывающие всё от кода до инфраструктуры для команд DevOps/SRE.

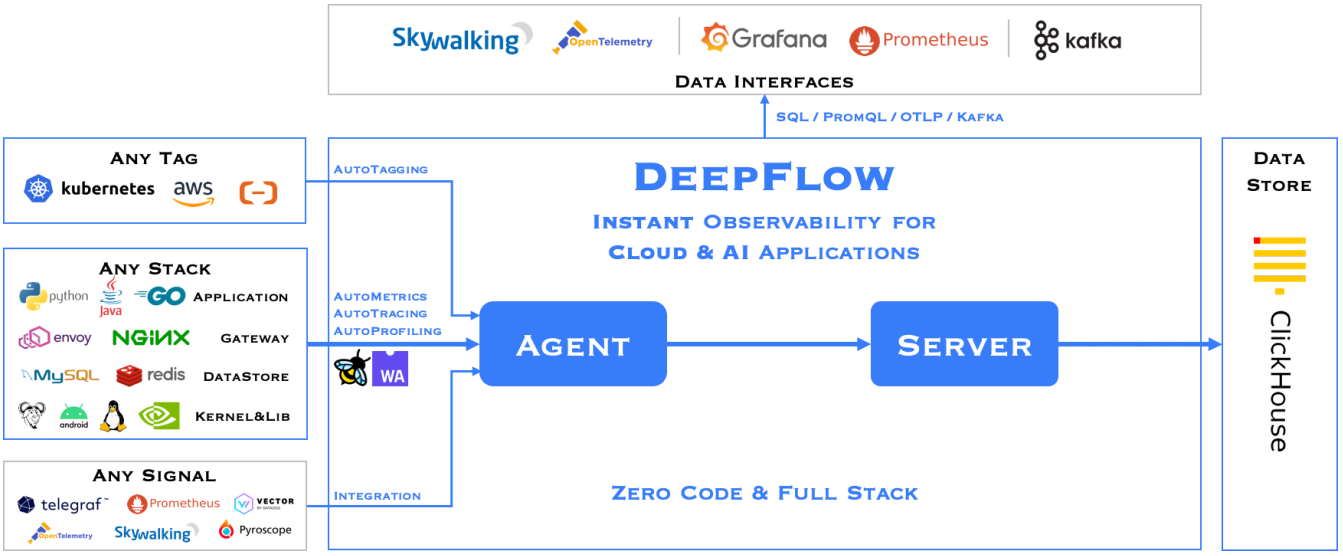
Использование технологии eBPF

Предполагая, что у вас есть базовое понимание eBPF, это безопасная и эффективная технология расширения функциональности ядра путём запуска программ в песочнице, что является революционным новшеством по сравнению с традиционными методами изменения исходного кода ядра и написания модулей ядра. Программы eBPF являются событийно-ориентированными, и когда ядро или пользовательские программы проходят через eBPF Hook, выполняется соответствующая программа eBPF, загруженная в точку Hook. Ядро Linux предопределяет ряд часто используемых точек Hook, а также можно динамически добавлять пользовательские точки Hook в ядро и приложения с помощью технологий kprobe и uprobe. Благодаря технологии Just-in-Time (JIT) эффективность выполнения кода eBPF может быть сопоставима с нативным кодом ядра и модулями ядра. Благодаря механизму Verification код eBPF выполняется безопасно, не вызывая сбоев ядра или бесконечных циклов.



Архитектура программного обеспечения

DeepFlow состоит из двух компонентов: Agent и Server. Агент запускается на каждом узле K8s, на устаревших хостах и облачных хостах, и отвечает за сбор данных AutoMetrics и AutoTracing всех процессов приложений на хосте. Server запускается в кластере K8s и предоставляет управление агентами, внедрение тегов, приём и запрос данных.



Установка DeepFlow

Введение

Требования к ядру

Возможности eBPF (AutoTracing, AutoProfiling) в DeepFlow имеют следующие требования к версии ядра:

Архитектура	Дистрибутив	Версия ядра	kprobe	Golang uprobe	OpenSSL uprobe	p
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

Архитектура	Дистрибутив	Версия ядра	kprobe	Golang uprobe	OpenSSL uprobe	p
		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
	Другие дистрибутивы	5.8+	Y	Y	Y	Y

Дополнительные замечания по версиям ядра:

1 В CentOS 7.9 и RedHat 7.6 были обратно портированы некоторые возможности eBPF (opens new window) в ядро 3.10. В этих двух дистрибутивах поддерживаемые DeepFlow версии ядра (зависимые точки hook) следующие:

- 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 Процессы Golang/OpenSSL внутри контейнеров не поддерживаются.

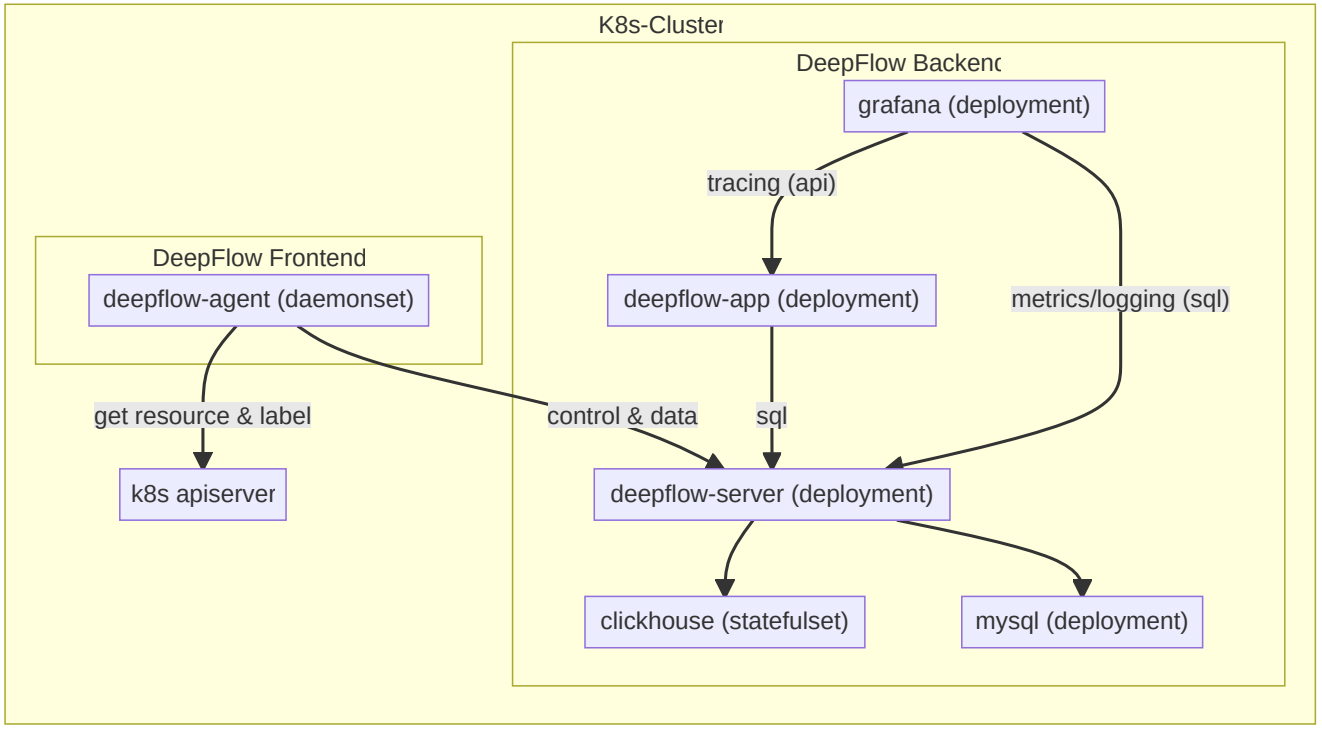
3 В версии ядра 4.14 к одному tracepoint не может быть прикреплено несколько программ eBPF (например, два и более deepflow-agent не могут работать одновременно), в других версиях такой проблемы нет.

ПРИМЕЧАНИЕ

Заявление RedHat:

> eBPF в Red Hat Enterprise Linux 7.6 предоставляется как Tech Preview и поэтому не имеет полной поддержки и не подходит для использования в продуктивной среде. Он предоставляется с основной целью расширения охвата и потенциального перехода к полной поддержке в будущем. eBPF в Red Hat Enterprise Linux 7.6 включён только для целей трассировки, что позволяет прикреплять программы eBPF к probe, tracepoint и perf событиям.

Топология развертывания



Подготовка

Storage Class

Рекомендуется использовать Persistent Volumes для хранения данных MySQL и ClickHouse, чтобы избежать ненужных затрат на обслуживание. Вы можете указать Storage Class по умолчанию или добавить параметр *--set global.storageClass=<your storageClass>*, чтобы выбрать Storage Class для создания PVC.

Для получения дополнительной информации о настройке хранилища обратитесь к документации по Storage.

Пакет

Загрузка пакета DeepFlow

Перейдите на Custom Portal для загрузки пакета DeepFlow.

Если у вас нет доступа к Custom Portal, обратитесь в техническую поддержку.

Загрузка пакета на платформу

Используйте инструмент violet для публикации пакета на платформу.

Подробные инструкции по использованию этого инструмента см. в разделе CLI.

Установка

Создайте ресурс приложения в пространстве имён *cpaas-system* для развертывания DeepFlow. Пример манифеста:

```yaml
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 Источник чарта DeepFlow. Если вы хотите использовать другой чарт, обратитесь в техническую поддержку.

2 Версия чарта DeepFlow. ДОЛЖНА совпадать с версией пакета DeepFlow, загруженного на платформу.

3 Storage Class, используемый для создания Persistent Volumes для MySQL и ClickHouse.

4 Имя кластера, в котором развернут DeepFlow.

5 Имя пользователя веб-интерфейса Grafana. Вы можете изменить его на желаемое.

6 Пароль веб-интерфейса Grafana. Рекомендуется изменить на надёжный пароль.

7 Корневой URL для веб-интерфейса Grafana. Вы можете изменить на желаемый URL.

8 Размер постоянного тома для MySQL. Настройте в соответствии с вашими потребностями.

9 Размер постоянного тома для пути ClickHouse. Настройте в соответствии с вашими потребностями.

10 Размер постоянного тома для пути хранения ClickHouse. Настройте в соответствии с вашими потребностями.

Дождитесь готовности приложения:

```
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
```

Настройка Ingress для веб-интерфейса Grafana

Создайте ресурс Ingress для доступа к веб-интерфейсу Grafana. Пример манифеста:

```yaml
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) Имя класса Ingress. Если вы устанавливаете DeepFlow в глобальный кластер, установите значение *global-alb2*. Также можно указать своё имя класса Ingress.

(2) Этот путь ДОЛЖЕН совпадать с корневым URL, настроенным в ресурсе Application.

Доступ к веб-интерфейсу Grafana

Вы можете получить доступ к веб-интерфейсу Grafana по URL, указанному в ресурсе Ingress, и войти, используя имя пользователя и пароль, заданные в ресурсе Application.

> **ВНИМАНИЕ**
>
> Настоятельно рекомендуется изменить пароль после первого входа.

Дополнительные ресурсы

- Мгновенная наблюдаемость для облачных и AI-приложений ↗
- eBPF — введение, учебные материалы и ресурсы сообщества ↗

Настройка правил ALB

Содержание

Введение

Что такое правило?

Правило — это Custom Resource (CR), который определяет, как входящие запросы сопоставляются и обрабатываются ALB.

Ingress, обрабатываемые ALB, могут быть автоматически преобразованы в правила.

Предварительные требования

установка alb и ft

Быстрая демонстрация правила

Ниже приведено демонстрационное правило, чтобы вы могли быстро получить представление о том, как использовать правила.

NOTE

правило должно быть привязано к frontend и alb через метки.

```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. Обязательно, указывает `Frontend`, к которому принадлежит это правило.
2. Обязательно, указывает ALB, к которому принадлежит это правило.
3. backendProtocol
4. certificate_name
5. dslx
6. Чем меньше число, тем выше приоритет.
7. serviceGroup

сопоставление запроса с dslx и приоритетом

dslx

DSLX — это предметно-ориентированный язык, используемый для описания критериев сопоставления. Например, правило ниже сопоставляет запрос, который удовлетворяет **всем** следующим условиям:

- url начинается с /app-a **или** /app-b
- метод — post
- параметр url с ключом group равен vip
- host соответствует *.app.com
- заголовок location равен east-1 или east-2
- есть cookie с именем uid
- исходные IP находятся в диапазоне 1.1.1.1-1.1.1.100

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

приоритет

Приоритет — это целое число от 0 до 10, где меньшие значения означают более высокий приоритет. Чтобы настроить приоритет правила в ingress, можно использовать следующий формат аннотации:

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

Для правил достаточно задать приоритет напрямую в `.spec.priority` целочисленным значением.

Действия

После того, как запрос сопоставлен с правилом, к запросу можно применить следующие действия

Функция	Описание	Ссылка
Timeout	Настройка параметров таймаута для запросов.	timeout
Redirect	Перенаправление входящих запросов на указанный URL.	redirect
CORS	Включение Cross-Origin Resource Sharing (CORS) для приложения.	cors
Header Modification	Позволяет изменять заголовки запроса или ответа.	header modification
URL Rewrite	Перезаписывает URL входящих запросов перед их пересылкой.	url-rewrite
WAF	Интеграция Web Application Firewall (WAF) для повышения безопасности.	waf
OTEL	Включение OpenTelemetry (OTEL) для распределённого трассирования и мониторинга.	otel
Keepalive	Включение или отключение функции keepalive для приложения.	keepalive

Backend

протокол backend

По умолчанию протокол backend установлен в HTTP. Если требуется использовать TLS повторное шифрование, можно настроить HTTPS.

Группа сервисов и политика сессионной аффинности

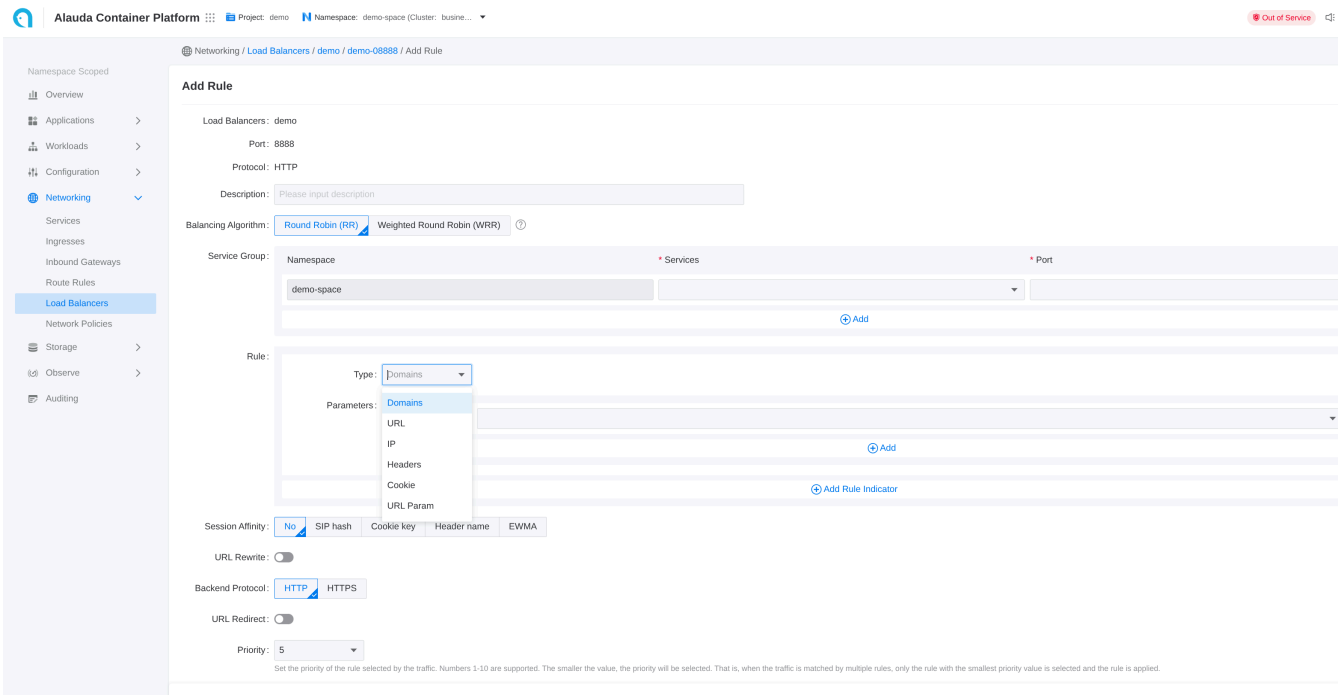
В правиле можно настроить один или несколько сервисов.

По умолчанию ALB использует алгоритм round-robin (RR) для распределения запросов между backend-сервисами. Однако можно назначать веса отдельным сервисам или выбрать другой алгоритм балансировки нагрузки.

Подробнее смотрите в разделе Balance Algorithm.

Создание правила

Использование веб-консоли



1. Перейдите в **Container Platform**.
2. В левой навигационной панели выберите **Network** > **Load Balancing**.
3. Кликните по имени балансировщика нагрузки.
4. Кликните по имени порта слушателя.
5. Нажмите **Add Rule**.
6. Используйте описания ниже для настройки соответствующих параметров.
7. Нажмите **Add**.

Каждое поле в веб-интерфейсе соответствует полю CR.

Использование CLI

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

HTTPS

Если протокол frontend (ft) — HTTPS или GRPCS, правило также может быть настроено на использование HTTPS. Сертификат можно указать либо в правиле, либо в ingress, чтобы сопоставить сертификат для конкретного порта.

Поддерживается termination, а также повторное шифрование, если протокол backend — HTTPS. Однако вы **не можете** указывать сертификат для связи с backend-сервисом.

Аннотация сертификата в Ingress

Сертификаты могут ссылаться на секреты из других namespace через аннотацию.

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

Сертификат в правиле

В `.spec.certificate_name` формат: `$secret_namespace/$secret_name`

Режим TLS

Edge Mode

В режиме edge клиент общается с ALB по HTTPS, а ALB общается с backend-сервисами по HTTP. Для этого:

1. создайте ft с протоколом https
2. создайте правило с backendProtocol http и укажите сертификат через `.spec.certificate_name`

Re-encrypt Mode

В режиме re-encrypt клиент общается с ALB по HTTPS, а ALB общается с backend-сервисами по HTTPS. Для этого:

1. создайте ft с протоколом https
2. создайте правило с backendProtocol https и укажите сертификат через `.spec.certificate_name`

Ingress

синхронизация ingress

Каждый ALB создаёт IngressClass с таким же именем и обрабатывает ingress в пределах одного проекта.

Если namespace ingress имеет метку вида `cpaas.io/project: demo`, это означает, что ingress принадлежит проекту `demo`.

ALB, у которых в конфигурации `.spec.config.projects` указан проект `demo`, автоматически преобразуют эти ingress в правила.

NOTE

ALB слушает ingress и автоматически создаёт `Frontend` или Rule. Поле `source` определяется следующим образом:

2.1. `spec.source.type` в настоящее время поддерживает только `ingress`.

2.2. `spec.source.name` — имя ingress.

2.3. `spec.source.namespace` — namespace ingress.

стратегия ssl

Для ingress без настроенных сертификатов ALB предоставляет стратегию использования сертификата по умолчанию.

Вы можете настроить Custom Resource ALB следующими параметрами:

- `.spec.config.defaultSSLStrategy`: определяет SSL-стратегию для ingress без сертификатов
- `.spec.config.defaultSSLCert`: задаёт сертификат по умолчанию в формате `$secret_ns/$secret_name`

Доступные SSL-стратегии:

- **Never**: не создавать правила на HTTPS-портах (поведение по умолчанию)
- **Always**: создавать правила на HTTPS-портах с использованием сертификата по умолчанию

Межкластерное соединение (Alpha)

Поддерживается настройка межкластерного соединения между кластерами с одинаковым сетевым режимом Kube-OVN, чтобы поды в кластерах могли взаимодействовать друг с другом. Cluster Interconnect Controller — это расширяемый компонент, предоставляемый Kube-OVN, который отвечает за сбор сетевой информации между разными кластерами и соединение сетей нескольких кластеров путем выдачи маршрутов.

Содержание

Предварительные требования

- CIDR подсетей разных кластеров не должны пересекаться.

- Необходим набор машин, доступных по IP для kube-ovn-controller каждого кластера, для развертывания контроллеров межкластерного соединения.
- Для каждого кластера должен существовать набор машин, доступных kube-ovn-controller по IP для межкластерного соединения, которые впоследствии будут использоваться в качестве шлюзовых узлов.
- Эта функция доступна только для VPC по умолчанию, пользовательские VPC не могут использовать функцию межсоединения.

Построен контроллер подключения Multi-node Kube-OVN

Доступны три метода развертывания: Deploy deployment (поддерживается в платформе с версии v3.16.0 и выше), Docker deployment и Containerd deployment.

Deploy Deployment

Примечание: Этот метод развертывания поддерживается в платформе с версии v3.16.0 и выше.

Шаги выполнения

1. Выполните следующую команду на Master-узле кластера, чтобы получить скрипт установки install-ic-server.sh.

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

2. Откройте скрипт в текущем каталоге и измените параметры следующим образом.

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

Измените параметры на следующие:

```
REGISTRY="<адрес репозитория образов Kube-OVN>"   ## Например:
REGISTRY="registry.alauda.cn:60080/acp/"
VERSION="<версия Kube-OVN>"   ## Например: VERSION="v1.9.25"
```

3. Сохраните скрипт и выполните его следующей командой.

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

Docker и Containerd Deployment

1. Выберите **три и более узлов в любом кластере** для развертывания Interconnected Controller. В этом примере подготовлено три узла.

2. Выберите любой узел в качестве Leader и выполните команды в соответствии с выбранным методом развертывания.

   Примечание: Перед настройкой проверьте наличие каталога ovn в `/etc`. Если его нет, создайте его командой `mkdir /etc/ovn`.

   - **Команды для Docker deployment**

     Примечание: Выполните команду `docker images | grep ovn`, чтобы получить адрес образа Kube-OVN.

     - Команда для Leader-узла:

```
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-адрес текущего узла>" \   ## Например: -e
LOCAL_IP="192.168.39.37"
-e NODE_IPS="<IP-адреса всех узлов через запятую>" \   ## Например: -e
NODE_IPS="192.168.39.22,192.168.39.24,192.168.39.37"
<адрес репозитория образов> bash start-ic-db.sh    ## Например:
192.168.39.10:60080/acp/kube-ovn:v1.8.8 bash start-ic-db.sh
```

- Команды для остальных двух узлов:

```
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-адрес текущего узла>" \   ## Например: -e
LOCAL_IP="192.168.39.24"
-e LEADER_IP="<IP-адрес Leader-узла>" \  ## Например: -e
LEADER_IP="192.168.39.37"
-e NODE_IPS="<IP-адреса всех узлов через запятую>" \   ## Например: -e
NODE_IPS="192.168.39.22,192.168.39.24,192.168.39.37"
<адрес репозитория образов> bash start-ic-db.sh   ## Например:
192.168.39.10:60080/acp/kube-ovn:v1.8.8  bash start-ic-db.sh
```

- **Команды для Containerd deployment**

Примечание: Выполните команду `crictl images | grep ovn`, чтобы получить адрес образа Kube-OVN.

- Команда для Leader-узла:

```
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-адреса всех узлов через запятую>" \   ## Например: --
env="NODE_IPS="192.168.178.97,192.168.181.93,192.168.177.192""
--env="LOCAL_IP=<IP-адрес текущего узла>" \   ## Например: --
env="LOCAL_IP="192.168.178.97""
<адрес репозитория образов> ovn-ic-db bash start-ic-db.sh   ## Например:
registry.alauda.cn:60080/acp/kube-ovn:v1.9.25 ovn-ic-db bash start-ic-db.sh
```

- Команды для остальных двух узлов:

```
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-адреса всех узлов через запятую>" \   ## Например: --
env="NODE_IPS="192.168.178.97,192.168.181.93,192.168.177.192"" \
--env="LOCAL_IP=<IP-адрес текущего узла>" \   ## Например: --
env="LOCAL_IP="192.168.181.93""
--env="LEADER_IP=<IP-адрес Leader-узла>" \   ## Например: --
env="LEADER_IP="192.168.178.97""
<адрес репозитория образов> ovn-ic-db bash start-ic-db.sh   ## Например:
registry.alauda.cn:60080/acp/kube-ovn:v1.9.25 ovn-ic-db bash start-ic-db.sh
```

Развертывание контроллера межкластерного соединения в глобальном кластере

На любом управляющем узле глобального кластера замените параметры согласно комментариям и выполните следующую команду для создания ресурса ConfigMap.

Примечание: Для корректной работы ConfigMap с именем ovn-ic в глобальном кластере не разрешается изменять. Если необходимо изменить параметры, удалите ConfigMap и корректно настройте его заново перед применением.

```
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"   # Адрес узлов, где расположен контроллер межкластерного соединения, в данном случае локальные IP трёх узлов с контроллером
  ic-nb-port: "6645"           # Порт nb контроллера межкластерного соединения, по умолчанию 6645
  ic-sb-port: "6646"           # Порт sb контроллера межкластерного соединения, по умолчанию 6646
EOF
```

Присоединение к межкластерному соединению

Добавление кластера с сетевым режимом Kube-OVN в межкластерное соединение.

Предварительные условия

Созданные подсети, **ovn-default** и **подсети для присоединения** в кластере не должны конфликтовать с сегментами других кластеров в группе межкластерного соединения.

Порядок действий

1. В левой навигационной панели нажмите **Clusters** > **Cluster of clusters**.
2. Нажмите на имя ***кластера***, который нужно добавить в межкластерное соединение.
3. В правом верхнем углу нажмите **Options** > **Cluster Interconnect**.
4. Нажмите **Join the cluster interconnect**.
5. Выберите шлюзовый узел для кластера.
6. Нажмите **Join**.

Соответствующие операции

Обновление информации о шлюзовом узле в межсоединённом кластере

Обновление информации о шлюзовых узлах кластера, которые уже присоединились к группе межкластерного соединения.

Порядок действий

1. В левой навигационной панели нажмите **Clusters** > **Cluster of clusters**.
2. Нажмите на ***имя кластера***, для которого нужно обновить информацию о шлюзовом узле.
3. В правом верхнем углу нажмите **Operations** > **Cluster Interconnect**.
4. Нажмите **Update Gateway Node** для кластера, информацию о шлюзовом узле которого нужно обновить.
5. Повторно выберите шлюзовый узел для кластера.
6. Нажмите **Update**.

Выход из межкластерного соединения

Кластер, который присоединился к группе межкластерного соединения, выходит из межкластерного соединения, при этом разрывается связь между подами этого кластера и подами внешних кластеров.

Порядок действий

1. В левой навигационной панели нажмите **Clusters** > **Cluster of clusters**.
2. Нажмите на имя ***кластера***, который нужно вывести из эксплуатации.
3. В правом верхнем углу нажмите **Options** > **Cluster Interconnect**.
4. Нажмите **Exit cluster interconnection** для кластера, из которого хотите выйти.
5. Введите имя кластера корректно.
6. Нажмите **Exit**.

Очистка остатков межсоединённого кластера

Если кластер удаляется без выхода из межкластерного соединения, на контроллере могут остаться некоторые остаточные данные. При попытке повторно создать кластер на этих узлах и присоединить его к межкластерному соединению могут возникать ошибки. Подробную информацию об ошибках можно посмотреть в логе `/var/log/ovn/ovn-ic.log` контроллера (kube-ovn-controller). Некоторые сообщения об ошибках могут содержать:

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

Шаги выполнения

1. Выйдите из межкластерного соединения для кластера, который нужно присоединить.
2. Выполните скрипт очистки в контейнере или pod.

   Скрипт очистки можно выполнить напрямую либо в контейнере ovn-ic-db, либо в pod ovn-ic-controller. Выберите один из следующих способов:

   Способ 1: Выполнение в контейнере ovn-ic-db

   - Войдите в контейнер ovn-ic-db и выполните очистку следующими командами.

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

Затем выполните одну из следующих команд очистки:

- Очистка по имени исходного кластера. Замените <*cluster-name*> на **имя исходного кластера**:

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

- Очистка по имени любого узла исходного кластера. Замените <*node-name*> на **имя любого узла исходного кластера**:

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

Способ 2: Выполнение в pod ovn-ic-controller

- Войдите в pod ovn-ic-controller и выполните очистку следующими командами.

```
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
```

Затем выполните одну из следующих команд очистки:

- Очистка по имени исходного кластера. Замените <*cluster-name*> на **имя исходного кластера**:

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

- Очистка по имени любого узла исходного кластера. Замените <*node-name*> на **имя любого узла исходного кластера**:

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

Удаление межсоединённого кластера

Примечание: Шаг 1 — Шаг 3 необходимо выполнить на всех **рабочих кластерах, присоединившихся к межсоединённому кластеру**.

Шаги выполнения

1. Выйдите из межкластерного соединения. Есть два способа выхода, выберите подходящий.

   - Удалите ConfigMap с именем ovn-ic-config в рабочем кластере командой:

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

   - Выйдите из межкластерного соединения через операции платформы.

2. Войдите в Leader Pod ovn-central следующей командой.

   ```
   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. Очистите логический коммутатор ts следующей командой.

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

4. Войдите на узел, где развернут контроллер, и удалите контроллер.

   - Команды для Docker:

     ```
     docker stop ovn-ic-db
     docker rm ovn-ic-db
     ```

   - Команды для Containerd:

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

5. Удалите ConfigMap с именем ovn-ic в глобальном кластере следующей командой.

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

Настройка высокой доступности шлюза кластера

Чтобы настроить высокодоступный шлюз кластера после присоединения к межкластерному соединению, выполните следующие шаги:

1. Войдите в кластер, который нужно преобразовать в высокодоступный шлюз, и выполните команду для изменения поля `enable-ic` на `false`.

   Примечание: Изменение поля `enable-ic` на `false` прервет межкластерное соединение до тех пор, пока оно снова не будет установлено в `true`.

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

2. Измените конфигурацию шлюзовых узлов, обновив поле `gw-nodes`, разделяя узлы английскими запятыми; также измените поле `enable-ic` на `true`.

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

   # Пример конфигурации
   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. Перейдите в Pod в кластере ovn-central и выполните команду `ovn-nbctl lrp-get-gateway-chassis {имя текущего кластера}-ts`, чтобы проверить, что конфигурация вступила в силу.

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

# Пример вывода. В данном случае значения 100 и 99 - это приоритеты, чем выше значение, тем выше приоритет соответствующего шлюзового узла.
docker-ts-71292a21-131d-492a-9f0c-0611af458950 100
docker-ts-1de7ee15-f372-4ab9-8c85-e54d61ea18f1 99
```

Endpoint Health Checker

Содержание

Overview

Endpoint Health Checker — это плагин кластера, предназначенный для мониторинга и управления состоянием здоровья сервисных эндпоинтов. Он автоматически удаляет нездоровые эндпоинты из балансировщиков нагрузки, чтобы трафик направлялся только на здоровые экземпляры, что повышает общую надежность и доступность сервиса.

Key Features

- **Автоматический мониторинг здоровья**: Непрерывно отслеживает состояние здоровья сервисных эндпоинтов
- **Интеграция с балансировщиком нагрузки**: Автоматически удаляет нездоровые эндпоинты из ротации балансировщика нагрузки
- **Доступность сервиса**: Обеспечивает направление трафика только на здоровые и доступные эндпоинты

Installation

Install via Marketplace

1. Перейдите в **Administrator** > **Marketplace** > **Cluster Plugins**.
2. Найдите в списке плагинов "**Alauda Container Platform Endpoint Health Checker**".
3. Нажмите **Install**, чтобы открыть страницу конфигурации установки.
4. Дождитесь, пока статус плагина не изменится на "**Ready**".

How It Works

Health Check Mechanism

Endpoint Health Checker — это специализированный компонент мониторинга здоровья, который гарантирует, что трафик направляется только на здоровые эндпоинты. Он работает, отслеживая сервисные эндпоинты и автоматически управляя их статусом доступности.

Core Functionality

Endpoint Health Checker работает следующим образом:

1. **Обнаружение сервисов**: Определяет сервисы и поды, настроенные для мониторинга здоровья

2. **Мониторинг здоровья подов**: Отслеживает состояние readiness и liveness probe подов, поддерживающих сервисные эндпоинты

3. **Активные проверки здоровья**: Выполняет активные проверки с использованием настраиваемых критериев:

- **Проверки TCP-соединения**: Устанавливает TCP-соединения для проверки доступности порта
- **Валидация HTTP/HTTPS-ответов**: Отправляет HTTP-запросы и проверяет коды ответов и содержимое

4. **Управление эндпоинтами**: Автоматически удаляет нездоровые эндпоинты из списков сервисных эндпоинтов, чтобы предотвратить маршрутизацию трафика на неработающие экземпляры

Health Check Process

Процесс проверки здоровья включает:

- **Интеграция с probe**: Использует результаты readiness и liveness probe Kubernetes как начальные индикаторы здоровья
- **Сетевое подключение**: Отправляет TCP или HTTP пакеты на порты целевых эндпоинтов для проверки доступности
- **Валидация ответа**: Оценивает статус ответа, время и содержимое для определения состояния эндпоинта
- **Автоматическое переключение**: Удаляет недоступные или неработающие эндпоинты из ротации балансировщика нагрузки

Activation Methods

Проверка здоровья может быть активирована двумя способами:

1. **Аннотация на уровне пода (рекомендуется)**:

   Добавьте аннотацию в шаблон пода в вашем 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. **readinessGates на уровне пода (устаревший способ)**:

   Настройте readinessGates в спецификации пода для старых версий:

```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
```

> **Примечание**: Конфигурация readinessGates относится к более старой версии. Рекомендуется использовать аннотацию пода `endpoint-health-checker.io/enabled: "true"` для новых развертываний.

Uninstallation

Чтобы удалить Endpoint Health Checker:

1. Перейдите в **Administrator** > **Marketplace** > **Cluster Plugins**.
2. Найдите установленный плагин "**Endpoint Health Checker**".
3. Нажмите меню опций и выберите **Uninstall**.
4. Подтвердите удаление при появлении запроса.

NodeLocal DNSCache

Содержание

Overview

NodeLocal DNSCache — это плагин кластера, который улучшает производительность DNS в кластере за счёт запуска прокси-кэша DNS на узлах кластера. Этот плагин снижает задержки DNS-запросов и повышает стабильность кластера, кэшируя DNS-ответы локально на каждом узле, минимизируя нагрузку на центральный DNS-сервис.

Key Features

- **Локальное кэширование DNS**: Кэширует DNS-ответы локально на каждом узле для снижения задержек запросов

- **Повышенная производительность**: Значительно сокращает время разрешения DNS для приложений

Important Notes

WARNING

Особенности развертывания:

1. **Режим Kube-OVN Underlay**: Плагин не поддерживает развертывание в режиме Kube-OVN Underlay. При развертывании в этом режиме возможны сбои DNS-запросов.
2. **Перезапуск kubelet**: Развертывание плагина приведёт к перезапуску kubelet.
3. **Требуется перезапуск Pod**: После успешного развертывания плагина он не повлияет на уже запущенные Pod, а вступит в силу только для новых Pod. При использовании CNI Kube-OVN необходимо вручную добавить параметр "--node-local-dns-ip=(IP-адрес локального DNS-кэша)" в kube-ovn-controller.
4. **Настройка NetworkPolicy**: Если в кластере настроен NetworkPolicy, необходимо дополнительно разрешить трафик в обоих направлениях для node CIDR и nodeLocalDNSIP в networkPolicy для обеспечения корректной связи.

Installation

Install via Marketplace

1. Перейдите в **Administrator** > **Marketplace** > **Cluster Plugins**.
2. Найдите в списке плагинов "**Alauda Build of NodeLocal DNSCache**".
3. Нажмите **Install** для открытия страницы конфигурации установки.
4. Настройте необходимые параметры:

Parameter	Description	Example Value
IP	IP-адрес сервера локального DNS-кэша на узле. Для IPv4 рекомендуется использовать адрес из диапазона 169.254.0.0/16, предпочтительно 169.254.20.10. Для IPv6 рекомендуется использовать адрес из диапазона fd00::/8, предпочтительно fd00::10.	169.254.20.10

5. Ознакомьтесь с заметками по развертыванию и убедитесь, что ваша среда соответствует требованиям.
6. Нажмите **Install** для завершения установки.
7. Дождитесь, пока статус плагина не изменится на "**Ready**".

How It Works

Architecture

```
Pod → NodeLocal DNSCache → [Cache Hit] → Pod
           ↓
    [Cache Miss] → CoreDNS → Response → Cache & Pod
```

Configuration

Network Policy Configuration

> **Важно**: Если в вашем кластере включён NetworkPolicy, необходимо настроить соответствующие правила для разрешения DNS-трафика к NodeLocal DNSCache. Без этих правил Pod могут не иметь возможности разрешать DNS-запросы.

При использовании NetworkPolicy убедитесь, что разрешён следующий DNS-трафик:

```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

Как сделать

Подготовка физической сети Kube-OVN Underlay

Подготовка физической сети Kube-OVN Underlay

Инструкции по использованию

Объяснение терминологии

Требования к среде

Пример конфигурации

Soft Data Center LB Solution (Alpha)

Soft Data Center LB Solution (Alpha)

Предварительные требования

Процедура

Проверка

Автоматическое взаимное подключение подсетей Underlay и Overlay

Автоматическое взаимное подключение подсетей Underlay и Overlay

Процедура

Установка Ingress-Nginx через Cluster Plugin

Задачи для Ingress-Nginx

ALB

Auth

Основные понятия

Быстрый старт

Связанные аннотации Ingress

forward-auth

basic-auth

CR

Специальная аннотация Ingress для ALB

Другие функции, связанные с Auth в Ingress-Nginx

Примечание: несовместимые моменты с Ingress-Nginx

Устранение неполадок

Развертывание высокодоступного VIP для ALB

Метод 1: Использование Service типа LoadBalancer для предоставления VIP

Метод 2: Использование внешнего ALB устройства для предоставления VIP

Модификация заголовков

Основные понятия

Примеры

HTTP Redirect

Основные понятия

CRD

Аннотация Ingress

Редирект на уровне порта

Редирект на уровне правила

L4/L7 Таймаут

Основные понятия

CRD

Что означает таймаут

Аннотации Ingress

Таймаут на уровне порта

ModSecurity

Терминология

Процедура эксплуатации

Связанные пояснения

Пример конфигурации

TCP/HTTP Keepalive

Основные понятия

CRD

Использование OAuth Proxy с ALB

Overview

Procedure

Result

Настройка GatewayApi Gateway через ALB

Привязка NIC в ALB

Принятие решений по выбору производительности ALB

Развертывание ALB

ALB

Порты слушателя (Frontend)

Логи и мониторинг

Проброс IPv6-трафика на IPv4-адреса внутри кластера через ALB

Метод настройки

Проверка результата

OTel

Терминология

Предварительные требования

Процедура

Связанные операции

Дополнительные сведения

Пример конфигурации

ALB Monitoring

Terminology

Procedure

Monitoring Metrics

CORS

Основные понятия

CRD

Политика сессионной аффинности балансировки нагрузки в ALB

Обзор

Доступные алгоритмы

Методы настройки

Рекомендуемые практики

Перезапись URL

Основные понятия

Конфигурация

Calico Network поддерживает шифрование WireGuard

Calico Network поддерживает шифрование WireGuard

Статус установки

Терминология

Примечания

Требования

Процедура

Проверка результата

Kube-OVN Overlay Network поддерживает шифрование IPsec

Kube-OVN Overlay Network поддерживает шифрование IPsec

Подготовка физической сети Kube-OVN Underlay

Сетевая инфраструктура контейнеров в режиме передачи Kube-OVN Underlay зависит от поддержки физической сети. Перед развертыванием сети Kube-OVN Underlay необходимо совместно с сетевым администратором спланировать и выполнить соответствующие настройки физической сети заранее, чтобы обеспечить сетевую связность.

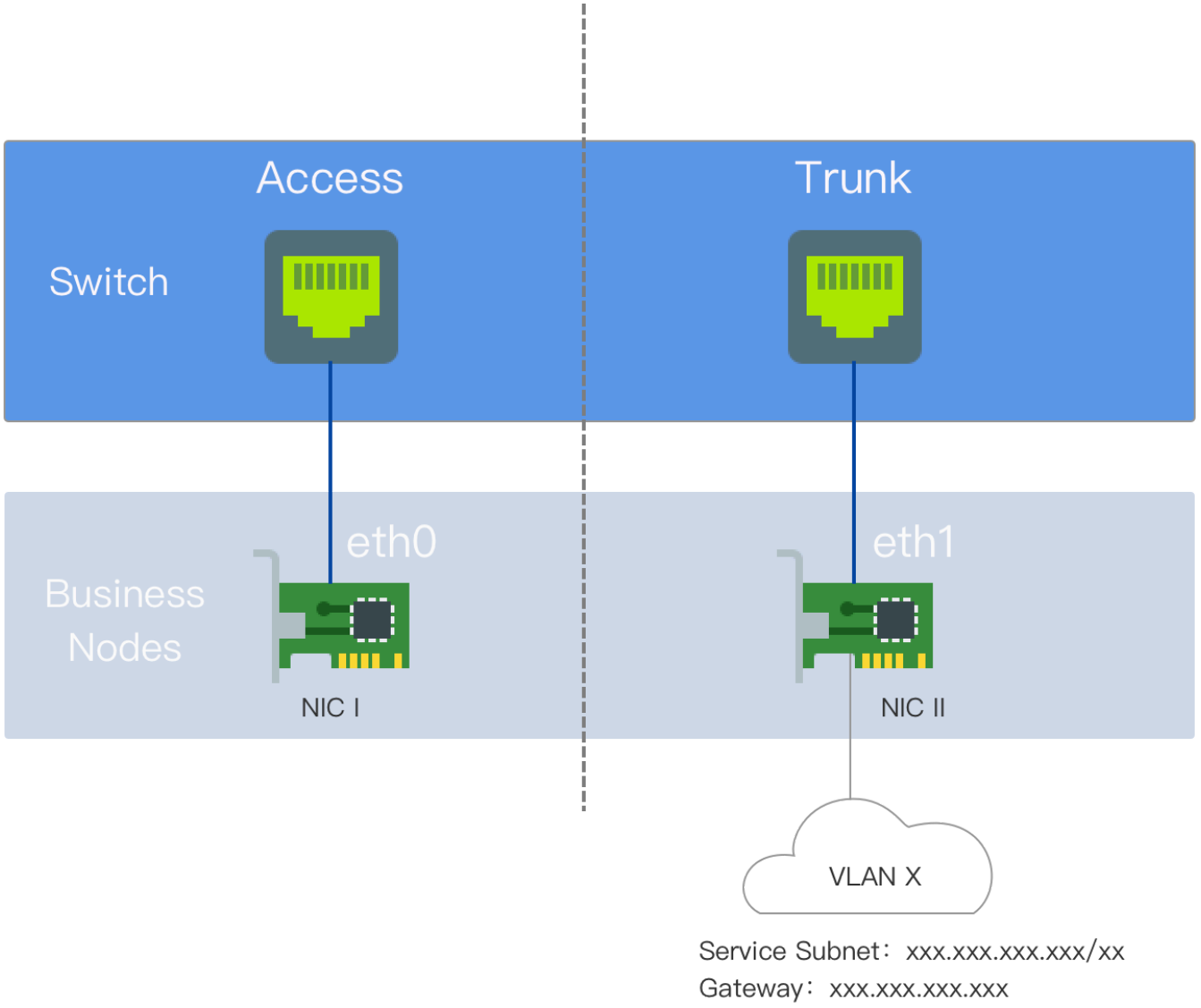
Содержание

Инструкции по использованию

Kube-OVN Underlay требует развертывания с несколькими сетевыми интерфейсными картами (NIC), при этом подсеть Underlay должна использовать исключительно один NIC. Другие типы трафика, например SSH, не должны использовать этот NIC и должны работать через другие NIC.

Перед использованием убедитесь, что на сервере узла имеется как минимум **двухсетевой NIC**, рекомендуется, чтобы скорость NIC была **не менее 10 Гбит/с или выше** (например, 10 Гбит/с, 25 Гбит/с, 40 Гбит/с).

- NIC Один: NIC с маршрутом по умолчанию, настроенный с IP-адресом, соединённый с внешним интерфейсом коммутатора, который настроен в режиме Access.
- NIC Два: NIC без маршрута по умолчанию и без настроенного IP-адреса, соединённый с внешним интерфейсом коммутатора, который настроен в режиме Trunk. Подсеть Underlay использует исключительно NIC Два.



Объяснение терминологии

VLAN (Virtual Local Area Network) — это технология, которая логически разделяет локальную сеть на несколько сегментов (или меньших LAN) для облегчения обмена

данными между виртуальными рабочими группами.

Появление технологии VLAN позволяет администраторам логически сегментировать различных пользователей в одной физической локальной сети на отдельные домены широковещания в соответствии с реальными потребностями приложений. Каждый VLAN состоит из группы рабочих станций с похожими требованиями и обладает теми же свойствами, что и физически сформированная LAN. Поскольку VLAN делятся логически, а не физически, рабочие станции в одном VLAN не ограничены одной физической областью; они могут находиться в разных физических сегментах LAN.

Основные преимущества VLAN включают:

- Сегментация портов. Даже на одном коммутаторе порты в разных VLAN не могут взаимодействовать друг с другом. Физический коммутатор может функционировать как несколько логических коммутаторов. Это часто используется для контроля взаимного доступа между разными отделами и площадками в сети.
- Безопасность сети. Разные VLAN не могут напрямую обмениваться данными, что исключает небезопасность широковещательной информации. Широковещательный и одноадресный трафик внутри VLAN не будет передаваться в другие VLAN, что помогает контролировать трафик, снижать затраты на оборудование, упрощать управление сетью и повышать безопасность сети.
- Гибкое управление. При изменении сетевой принадлежности пользователя нет необходимости менять порты или кабели; достаточно изменить конфигурацию программно.

Требования к среде

В режиме Underlay Kube-OVN связывает физический NIC с OVS и отправляет пакеты напрямую во внешнюю сеть через этот физический NIC. Возможности L2/L3 пересылки зависят от базовых сетевых устройств. Соответствующие шлюз, VLAN и политики безопасности должны быть предварительно настроены на базовых сетевых устройствах.

- **Требования к сетевой конфигурации**

- Kube-OVN проверяет доступность шлюза с помощью протокола ICMP при запуске контейнеров; базовый шлюз должен отвечать на ICMP-запросы.
- Для трафика доступа к сервисам Pods сначала отправляют пакеты на шлюз, который должен иметь возможность пересылать пакеты обратно в локальную подсеть.
- Если на коммутаторе или мосту включена функция Hairpin, **Hairpin должен быть отключён**. При использовании среды виртуальных машин VMware необходимо установить **Net.ReversePathFwdCheckPromisc** на хосте VMware в значение **1**, тогда Hairpin отключать не нужно.
- NIC, используемый для мостирования, **не может** быть **Linux Bridge**.
- Режимы агрегации NIC поддерживают Mode 0 (balance-rr), Mode 1 (active-backup), Mode 4 (802.3ad), Mode 6 (balance-alb), рекомендуется использовать 0 или 1. Другие режимы агрегации не тестировались, используйте их с осторожностью.

- **Требования к конфигурации слоя IaaS (виртуализации)**
  - Для среды виртуальных машин OpenStack необходимо отключить **PortSecurity** для соответствующего сетевого порта.
  - Для сети vSwitch VMware параметры **MAC Address Changes**, **Forged Transmits** и **Promiscuous Mode Operation** должны быть установлены в **Accept**.
  - Для публичных облаков, таких как AWS, GCE и Alibaba Cloud, сети в режиме Underlay не поддерживаются из-за отсутствия возможности задания MAC-адресов пользователем.

Пример конфигурации

В этом примере узлы — физические машины с двумя NIC. NIC Один — это NIC с маршрутом по умолчанию; NIC Два — NIC без маршрута по умолчанию и без настроенного IP-адреса, используемый исключительно для подсети Underlay. NIC Два соединён с внешним коммутатором.

- На стороне коммутатора интерфейс, подключённый к NIC Два, должен быть настроен в режиме Trunk с разрешением соответствующих VLAN.
- Настройте адрес шлюза подсети кластера на соответствующем интерфейсе vlan-interface. При необходимости двойного стека можно одновременно настроить IPv6-адрес шлюза.
- Если шлюз находится за файрволом, необходимо разрешить доступ от узлов к сети cluster-cidr.
- Конфигурация NIC сервера не требуется.

Конфигурация коммутатора

Настройка VLAN интерфейса:

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

Настройка интерфейса, подключённого к NIC Два:

```
#
interface Ten-GigabitEthernet1/0/19
  port link mode bridge
  port link-type trunk  // Настройка интерфейса в режим Trunk
  undo port trunk permit vlan 1
  port trunk permit vlan 74  // Разрешение прохождения соответствующего VLAN
#
```

Проверка сетевой связности

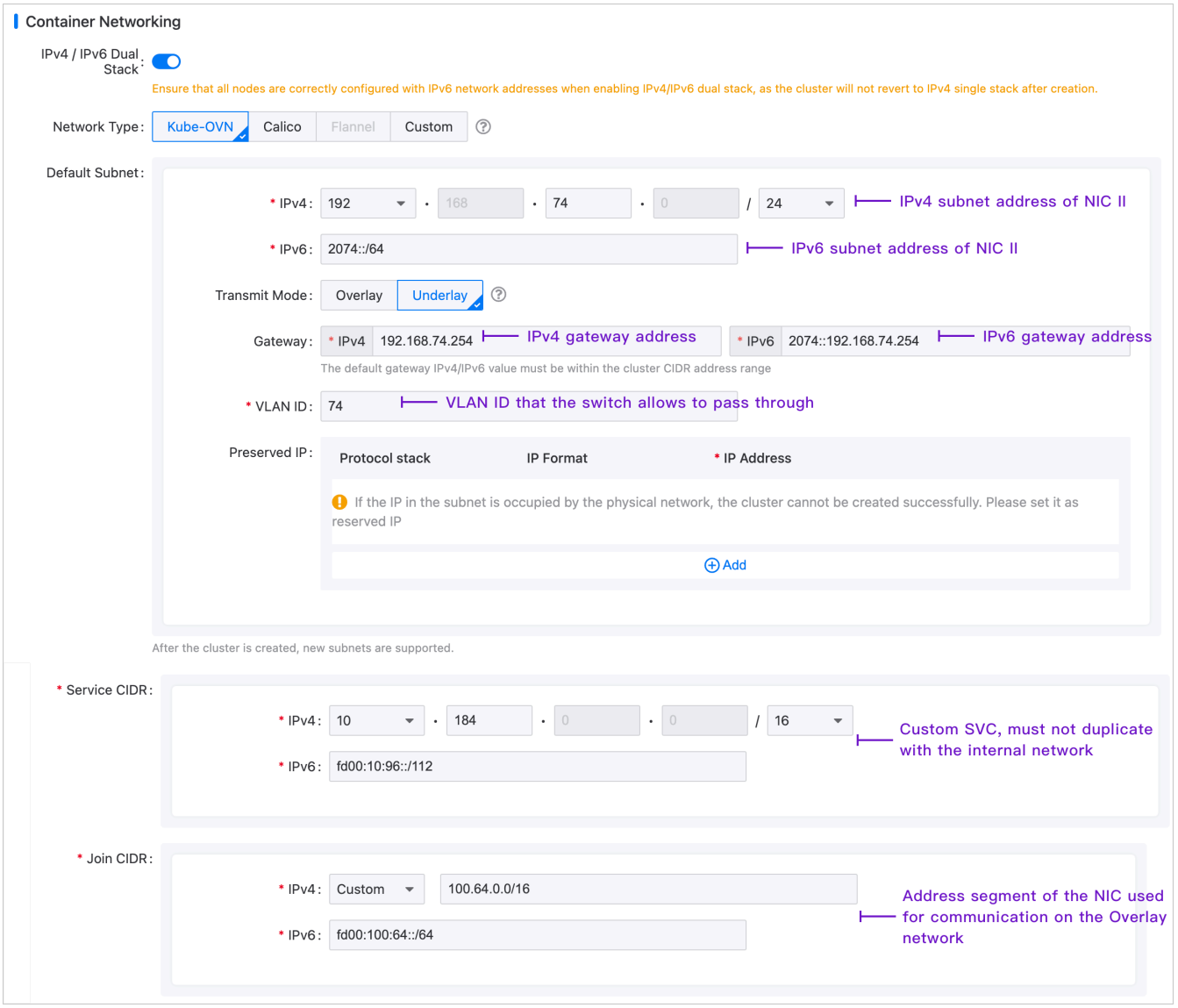
Проверьте, может ли NIC Два связаться с адресом шлюза:

```
ip link add ens224.74 link ens224 type vlan id 74  // Имя NIC – ens224, VLAN ID – 74
ip link set ens224.74 up
ip addr add 192.168.74.200/24 dev ens224.74  // Выберите тестовый адрес из подсети Underlay, здесь 192.168.74.200/24
ping 192.168.74.254  // Если пинг до шлюза успешен, значит физическая среда соответствует требованиям развертывания
ip addr del 192.168.74.200/24 dev ens224.74  // Удалите тестовый адрес после проверки
ip link del ens224.74  // Удалите подинтерфейс после проверки
```

Конфигурация платформы

В левой навигационной панели нажмите **Cluster Management > Cluster**, затем нажмите **Create Cluster**. Для подробной процедуры настройки обратитесь к документу Create Cluster, где показана конфигурация сетей контейнеров на изображении ниже.

Примечание: Подсеть Join не имеет практического значения в среде Underlay и служит в основном для последующего создания подсети Overlay, предоставляя диапазон IP-адресов, необходимый для связи между узлами и группами контейнеров.


Container Networking
IPv4 / IPv6 Dual Stack:
Ensure that all nodes are correctly configured with IPv6 network addresses when enabling IPv4/IPv6 dual stack, as the cluster will not revert to IPv4 single stack after creation.
Network Type: Kube-OVN Calico Flannel Custom
Default Subnet:
* IPv4: 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
Custom SVC, must not duplicate with the internal network
* IPv6: fd00:10:96::/112
* Join CIDR:
* IPv4: Custom 100.64.0.0/16
Address segment of the NIC used for communication on the Overlay network
* IPv6: fd00:100:64::/64

Soft Data Center LB Solution (Alpha)

Разверните чисто программный балансировщик нагрузки (LB) для дата-центра, создав высокодоступный балансировщик нагрузки вне кластера, обеспечивающий балансировку нагрузки для нескольких ALB с целью стабильной работы бизнес-приложений. Поддерживается настройка только IPv4, только IPv6 или двойного стека IPv4 и IPv6.

Содержание

Предварительные требования

1. Подготовьте два или более хост-узлов в качестве LB. Рекомендуется установить на узлах LB операционную систему Ubuntu 22.04 для сокращения времени перенаправления трафика LB на аномальные backend-узлы.
2. Предварительно установите следующее программное обеспечение на все хост-узлы внешнего LB (в этом разделе в качестве примера рассматриваются два хост-узла внешнего LB):
   - `ipvsadm`
   - `Docker (20.10.7)`
3. Убедитесь, что служба Docker настроена на автоматический запуск при загрузке на каждом хосте с помощью команды: `sudo systemctl enable docker.service`.

4. Убедитесь, что часы каждого хост-узла синхронизированы.

5. Подготовьте образ Keepalived, используемый для запуска службы внешнего LB; платформа уже содержит этот образ. Адрес образа имеет следующий формат: `<image repository address>/tkestack/keepalived:<version suffix>`. Суффикс версии может незначительно отличаться в разных версиях. Вы можете получить адрес репозитория образов и суффикс версии следующим образом. В данном документе в качестве примера используется `build-harbor.alauda.cn/tkestack/keepalived:v3.16.0-beta.3.g598ce923`.

   - В глобальном кластере выполните `kubectl get prdb base -o json | jq .spec.registry.address`, чтобы получить параметр **image repository address**.
   - В каталоге, где распакован установочный пакет, выполните `cat ./installer/res/artifacts.json |grep keepalived -C 2|grep tag|awk '{print $2}'|awk -F '"' '{print $2}'`, чтобы получить **version suffix**.

Процедура

Примечание: Следующие операции необходимо выполнить один раз на каждом хост-узле внешнего LB, при этом `hostname` хост-узлов не должен дублироваться.

1. Добавьте следующую конфигурацию в файл `/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. Добавьте следующую конфигурацию в файл `/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. Перезагрузите систему командой `reboot`.

4. Создайте папку для конфигурационного файла Keepalived.

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

5. Измените параметры конфигурации согласно комментариям в следующем файле и сохраните его в папке `/etc/keepalived/` под именем `alive.yaml`.

```
instances:
  - vip: # Можно настроить несколько VIP
      vip: 192.168.128.118 # VIP должны быть уникальными
      id: 20 # ID каждого VIP должен быть уникальным, опционально
      interface: "eth0"
      check_interval: 1 # опционально, по умолчанию 1: интервал выполнения
скрипта проверки
      check_timeout: 3  # опционально, по умолчанию 3: таймаут скрипта проверки
      name: "vip-1" # Идентификатор экземпляра, может содержать только
буквенно-цифровые символы и дефисы, не может начинаться с дефиса
      peer: [ "192.168.128.116", "192.168.128.75" ] # IP узлов Keepalived, в
сгенерированном keepalived.conf все IP интерфейса будут удалены
https://github.com/osixia/docker-keepalived/issues/33
      kube_lock:
        kubecfgs: # Список kube-config, используемый kube-lock для
последовательного выбора лидера в Keepalived
          - "/live/cfg/kubecfg/kubecfg01.conf"
          - "/live/cfg/kubecfg/kubecfg02.conf"
          - "/live/cfg/kubecfg/kubecfg03.conf"
    ipvs: # Конфигурация для опции IPVS
      ips: [ "192.168.143.192", "192.168.138.100","192.168.129.100" ] # IPVS backend,
замените IP узлов k8s master на IP узлов ALB
      ports: # Настройка логики проверки здоровья для каждого порта VIP
        - port: 80 # Порт виртуального сервера должен совпадать с портом
реального сервера
          virtual_server_config: |
            delay_loop 10  # Интервал выполнения проверки здоровья реального
сервера
            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: # Список kube-config, используемый kube-lock для
```

```
последовательного выбора лидера в 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. Выполните следующую команду в бизнес-кластере для проверки срока действия сертификата в конфигурационном файле, чтобы убедиться, что сертификат еще действителен. После истечения срока действия сертификата функциональность LB станет недоступной, потребуется обратиться к администратору платформы для обновления сертификата.

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

7. Скопируйте файл `/etc/kubernetes/admin.conf` с трех Master-узлов Kubernetes-кластера в папку `/etc/keepalived/kubecfg` на внешних LB-узлах, присвоив им имена с индексом, например, `kubecfg01.conf`, и измените в этих трех файлах адреса узлов `apiserver` на реальные адреса узлов Kubernetes-кластера.

   Примечание: После обновления сертификата платформы этот шаг необходимо повторить, перезаписав исходные файлы.

8. Проверьте действительность сертификатов.

   1. Скопируйте `/usr/bin/kubectl` с Master-узла бизнес-кластера на узел LB.

   2. Выполните `chmod +x /usr/bin/kubectl` для предоставления прав на выполнение.

3. Выполните следующие команды для подтверждения действительности сертификата.

```
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
```

Если возвращаются следующие результаты, сертификат действителен.

```
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. Загрузите образ Keepalived на внешний LB-узел и запустите Keepalived с помощью 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. На узле, с которого осуществляется доступ к `keepalived`, выполните команду: `sysctl -w net.ipv4.conf.all.arp_accept=1`.

Проверка

1. Выполните команду `ipvsadm -ln` для просмотра правил IPVS, вы увидите правила IPv4 и IPv6, применимые к ALB бизнес-кластера.

```
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. Выключите LB-узел, на котором расположен VIP, и проверьте, успешно ли VIP для IPv4 и IPv6 мигрирует на другой узел, обычно это происходит в течение 20 секунд.

3. С помощью команды `curl` на узле, не являющемся LB, проверьте, нормальна ли связь с VIP.

```
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>
```

Автоматическое взаимное подключение подсетей Underlay и Overlay

Если в кластере присутствуют подсети как Underlay, так и Overlay, по умолчанию Pods в подсети Overlay могут обращаться к IP-адресам Pods в подсети Underlay через шлюз с использованием NAT. Однако Pods в подсети Underlay для доступа к Pods в подсети Overlay необходимо настроить маршрутизацию на узлах.

Для автоматического взаимного подключения подсетей Underlay и Overlay можно вручную изменить YAML-файл подсети Underlay. После настройки Kube-OVN также будет использовать дополнительный IP Underlay для подключения подсети Underlay и логического маршрутизатора ovn-cluster, устанавливая соответствующие правила маршрутизации для обеспечения взаимосвязи.

Содержание

Процедура

1. Перейдите в раздел **Administrator**.
2. В левой навигационной панели нажмите **Cluster Management** > **Resource Management**.
3. Введите **Subnet** для фильтрации объектов ресурсов.
4. Нажмите на ⋮ > **Update** рядом с подсетью Underlay, которую необходимо изменить.
5. Измените YAML-файл, добавив поле `u2oInterconnection: true` в раздел `Spec`.

6. Нажмите **Update**.

Примечание: Существующие вычислительные компоненты в подсети Underlay необходимо пересоздать, чтобы изменения вступили в силу.

Установка Ingress-Nginx через Cluster Plugin

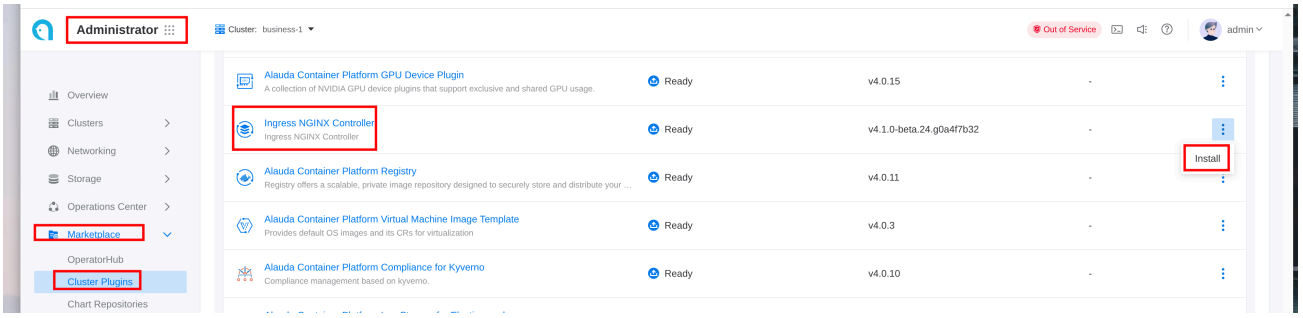
Содержание

Overview

NGINX Ingress Controller разворачивается как cluster plugin в пространстве имён `cpaas-system`. В этом руководстве описывается установка, настройка и лучшие практики управления Ingress NGINX Controller в вашем Kubernetes кластере.

Installation

1. Перейдите в `Administrator -> Marketplace -> Cluster Plugin`
2. Найдите плагин Ingress NGINX и нажмите `Install`



Configuration Management

Updating Configuration

1. Настройте kubectl для использования **global** контекста кластера
2. Получите имя ModuleInfo для вашего кластера:

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

3. Отредактируйте конфигурацию ModuleInfo:

```
kubectl edit minfo $MINFO
```

Раздел `spec.config` соответствует значениям Helm chart Ingress NGINX.

Common Configuration Scenarios

Exposing via LoadBalancer

Для экспонирования Ingress Controller через сервис LoadBalancer:

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

MetalLB Integration

Для указания VIP LoadBalancer при использовании MetalLB:

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

Advanced Controller Deployment Settings

Настройка сетевого режима, количества реплик, лимитов ресурсов и выбора узлов:

```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

Включение функционала SSL passthrough:

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

IPv6 Single-Stack Support

Настройка для среды с поддержкой только IPv6:

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

Performance Tuning

Resource Allocation Guidelines

Small Scale (< 300 QPS)

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

Medium Scale (< 10,000 QPS)

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

Large Scale (< 20,000 QPS)

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

High Performance (Unlimited)

Для максимальной производительности без ограничений по CPU:

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

Important Notes

Limitations

- Поддерживается только один экземпляр Ingress NGINX Controller на кластер

Version Compatibility

Текущая карта версий:

ACP Ingress-NGINX 4.1.x соответствует официальному chart 4.12.2 (controller v1.12.3)

Related Resources

tasks for ingress-nginx

official Ingress NGINX chart ↗

official Ingress NGINX documentation ↗

Задачи для Ingress-Nginx

Содержание

Предварительные требования

установка ingress-nginx

Максимальное количество соединений

max-worker-connections ↗

Таймаут

настройка таймаута ↗

Сессии с сохранением состояния (Sticky Session)

настройка sticky session ↗

Модификация заголовков

Действие	Ссылка
установка заголовка в запросе	proxy-set-header ↗
удаление заголовка в запросе	установка пустого заголовка в запросе
установка заголовка в ответе	configuration-snippets ↗ с директивой more-set-header ↗
удаление заголовка в ответе	hide-headers ↗

Перезапись URL

rewrite ↗

HSTS (HTTP Strict Transport Security)

настройка HSTS ↗

Ограничение скорости (Rate Limiting)

настройка ограничения скорости ↗

WAF

modsecurity ↗

Управление заголовком Forward

x-forwarded-prefix-header ↗

HTTPS

TLS повторное шифрование и проверка сертификата backend

проверка сертификата backend по HTTPS ↗

TLS терминация на краю

backend protocol ↗

Passthrough

ssl-passthrough ↗

Сертификат по умолчанию

default-ssl-certificate ↗

Menu

ALB

Auth

Основные понятия

Быстрый старт

Связанные аннотации Ingress

forward-auth

basic-auth

CR

Специальная аннотация Ingress для ALB

Другие функции, связанные с Auth в Ingress-Nginx

Примечание: несовместимые моменты с Ingress-Nginx

Устранение неполадок

Развертывание высокодоступного VIP для ALB

Метод 1: Использование Service типа LoadBalancer для предоставления VIP

Метод 2: Использование внешнего ALB устройства для предоставления VIP

Модификация заголовков

Основные понятия

Примеры

HTTP Redirect

Основные понятия

CRD

Аннотация Ingress

Редирект на уровне порта

Редирект на уровне правила

L4/L7 Таймаут

Основные понятия

CRD

Что означает таймаут

Аннотации Ingress

Таймаут на уровне порта

ModSecurity

Терминология

Процедура эксплуатации

Связанные пояснения

Пример конфигурации

TCP/HTTP Keepalive

Основные понятия

CRD

Использование OAuth Proxy с ALB

Настройка GatewayApi Gateway через ALB

Привязка NIC в ALB

Принятие решений по выбору производительности ALB

Малое производственное окружение

Среднее производственное окружение

Большое производственное окружение

Рекомендации по развертыванию в особых сценариях

Выбор режима использования балансировщика нагрузки

Развертывание ALB

ALB

Порты слушателя (Frontend)

Логи и мониторинг

Проброс IPv6-трафика на IPv4-адреса внутри кластера через ALB

Метод настройки

Проверка результата

OTel

Терминология

Предварительные требования

Процедура

Связанные операции

Дополнительные сведения

Пример конфигурации

ALB Monitoring

Terminology

Procedure

Monitoring Metrics

CORS

Основные понятия

CRD

Политика сессионной аффинности балансировки нагрузки в ALB

Обзор

Доступные алгоритмы

Методы настройки

Рекомендуемые практики

Перезапись URL

Основные понятия

Конфигурация

Auth

Содержание

Основные понятия

Что такое Auth

Auth — это механизм, который выполняет аутентификацию до того, как запрос достигнет фактического сервиса. Он позволяет обрабатывать аутентификацию на уровне ALB единообразно, без необходимости реализовывать логику аутентификации в каждом backend-сервисе.

Поддерживаемые методы аутентификации

ALB поддерживает два основных метода аутентификации:

1. **Forward Auth (Внешняя аутентификация)**
   - Отправка запроса во внешний сервис аутентификации для проверки личности пользователя
   - Сценарии применения: требуется сложная логика аутентификации, например OAuth, SSO и т.д.
   - Рабочий процесс:

1. Запрос пользователя поступает на ALB
2. ALB пересылает информацию об аутентификации в сервис аутентификации
3. Сервис аутентификации возвращает результат проверки
4. На основе результата аутентификации принимается решение о допуске к backend-сервису

2. **Basic Auth (Базовая аутентификация)**

- Простой механизм аутентификации на основе имени пользователя и пароля
- Сценарии применения: простое управление доступом, защита среды разработки
- Рабочий процесс:

1. Запрос пользователя поступает на ALB
2. ALB проверяет имя пользователя и пароль в запросе
3. Сравнивает с настроенной информацией для аутентификации
4. Если проверка успешна, запрос пересылается в backend-сервис

Способы настройки Auth

1. **Глобальный Auth**

- Настраивается на уровне ALB, применяется ко всем сервисам
- Конфигурируется в ALB или FT CR

2. **Auth на уровне пути**

- Настраивается на конкретном пути Ingress
- Конфигурируется на конкретном Rule
- Может переопределять глобальную конфигурацию auth

3. **Отключение Auth**

- Отключение аутентификации для конкретного пути
- Конфигурируется в Ingress с аннотацией: `alb.ingress.cpaas.io/auth-enable: "false"`
- Конфигурируется в Rule с помощью CR

Обработка результата Auth

- Успешная аутентификация: запрос будет переслан в backend-сервис
- Ошибка аутентификации: возвращается ошибка 401 unauthorized
- Можно настроить поведение перенаправления после ошибки аутентификации (применимо к Forward Auth)

Быстрый старт

Настройка Basic Auth с ALB

Развертывание 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
```

Настройка Secret и 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
```

Проверка

```
# echo "Zm9vOiJhYXIi" | base64 -d # foo:bar
curl -v -X GET -H "Authorization: Basic Zm9vOmJhcg==" http://$ALB_IP:80/app-file #
должен вернуть 200
# неправильный пароль
curl -v -X GET -H "Authorization: Basic XXXXOmJhcg==" http://$ALB_IP:80/app-file #
должен вернуть 401
```

Связанные аннотации Ingress

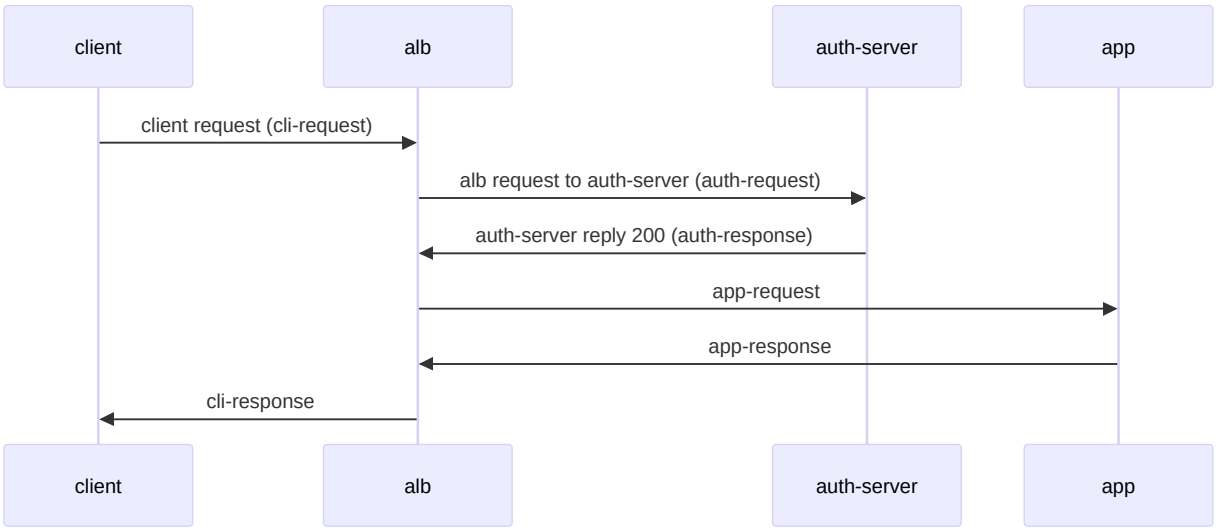
Ingress-nginx определяет ряд аннотаций для настройки конкретных деталей процесса аутентификации. Ниже приведён список аннотаций, поддерживаемых ALB, где "v" означает поддержку, а "x" — отсутствие поддержки.

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

	support	type	note
nginx.ingress.kubernetes.io/auth-request-redirect	v	string	
nginx.ingress.kubernetes.io/auth-always-set-cookie	v	boolean	
nginx.ingress.kubernetes.io/auth-snippet	x	string	
basic-auth			аутентификация по имени пользователя и паролю с использованием секрета
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: поддерживается apr1 **digest: не поддерживается**
auth-cache			
nginx.ingress.kubernetes.io/auth-cache-key	x	string	
nginx.ingress.kubernetes.io/auth-cache-duration	x	string	
auth-keepalive			keepalive при отправке запроса.

	support	type	note
			Поведение keepalive задаётся через набор аннотаций
nginx.ingress.kubernetes.io/auth-keepalive	x	number	
nginx.ingress.kubernetes.io/auth-keepalive-share-vars	x	"true" or "false"	
nginx.ingress.kubernetes.io/auth-keepalive-requests	x	number	
nginx.ingress.kubernetes.io/auth-keepalive-timeout	x	number	
auth-tls ↗			при https-запросе дополнительная проверка сертификата
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



Связанные аннотации:

- 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

Эти аннотации описывают изменения, вносимые в auth-request, app-request и cli-response в приведённой выше диаграмме.

Конструкция связанных аннотаций

auth-url

URL для auth-request, значение может быть переменной.

auth-method

Метод для auth-request.

auth-proxy-set-headers

Значение — ссылка на ConfigMap в формате `ns/name`. По умолчанию все заголовки из cli-request отправляются auth-server. Дополнительные заголовки можно настроить через proxy_set_header. По умолчанию отправляются следующие заголовки:

```
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;
```

Конструкция аннотаций, связанных с app-request

auth-response-headers

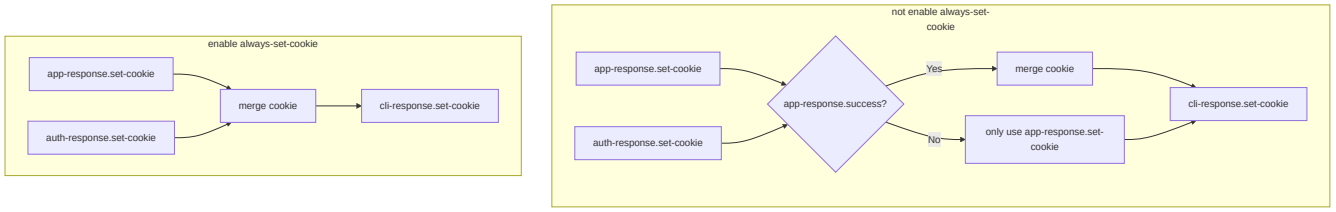
Значение — строка с разделёнными запятыми заголовками, позволяющая передавать определённые заголовки из auth-response в app-request. пример:

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

Когда ALB инициирует app-request, он включит Remote-User и Remote-Name из заголовков auth-response.

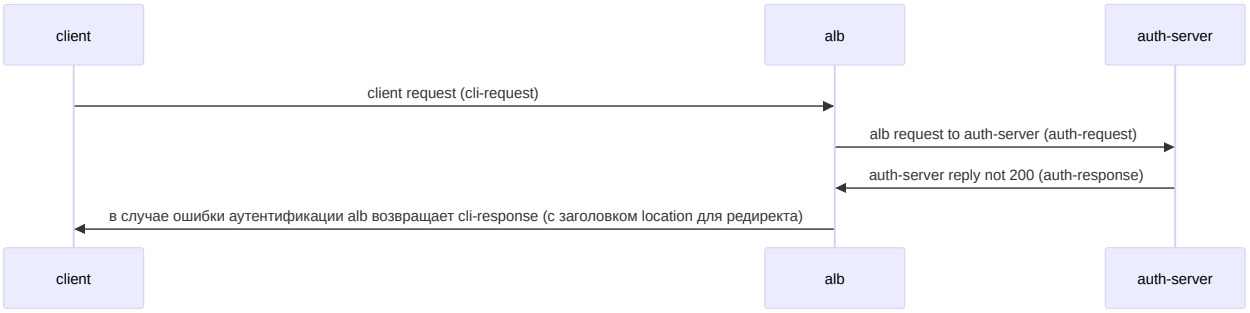
Обработка cookie

auth-response и app-response могут устанавливать cookie. По умолчанию, только если app-response успешен, cookie из auth-response.set-cookie будут объединены с cli-response.set-cookie.



Конфигурация, связанная с Redirect sign

Когда auth-server возвращает 401, можно установить заголовок redirect в cli-response, чтобы браузер перенаправился на url, указанный в auth-signin для прохождения проверки.



auth-signin

Значение — url, указывает заголовок location в cli-response.

auth-signin-redirect-param

Имя параметра запроса в signin-url, по умолчанию rd. Если signin-url не содержит параметр с именем, указанным в `auth-signin-redirect-param`, alb автоматически добавит этот параметр. Значение параметра будет установлено в `$pass_access_scheme://$http_host$escaped_request_uri`, используется для записи исходного URL запроса.

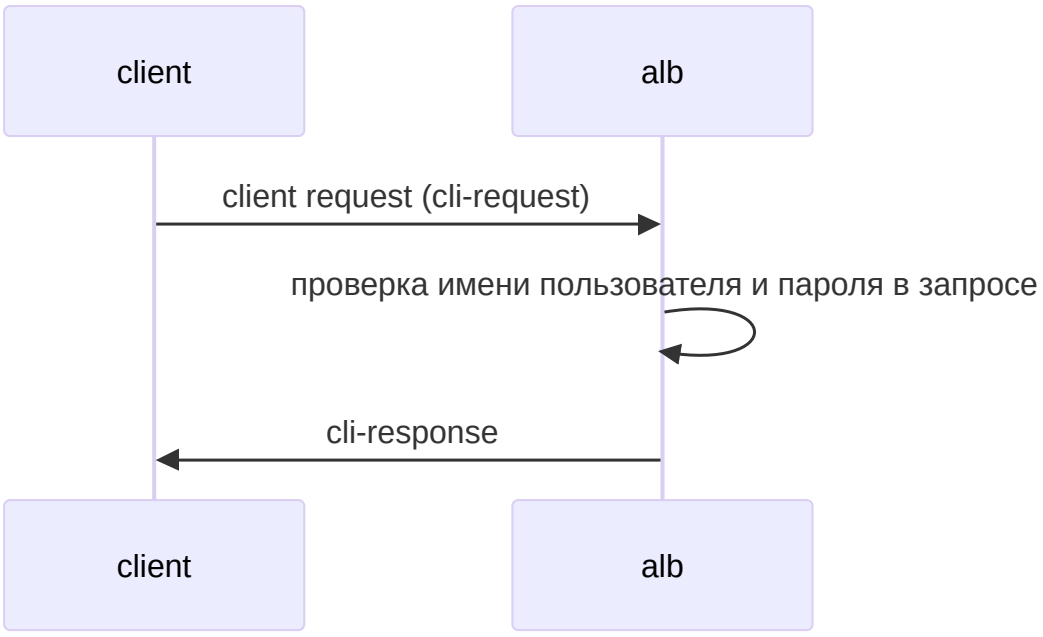
auth-request-redirect

Устанавливает заголовок `x-auth-request-redirect` в auth-request.

---

basic-auth

basic-auth — процесс аутентификации, описанный в RFC 7617↗. Процесс взаимодействия следующий:



auth-realm

описание защищённой области↗ Это значение realm в заголовке `WWW-Authenticate` cli-response. WWW-Authenticate: Basic realm="$realm"

auth-type

Тип схемы аутентификации, в настоящее время поддерживается только basic

auth-secret

Ссылка на секрет с именем пользователя и паролем, формат ns/name

auth-secret-type

Секрет поддерживает два типа:

1. auth-file: данные секрета содержат только один ключ "auth", значение — строка в формате Apache htpasswd. Например:

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

2. auth-map: в данных секрета каждый ключ — это имя пользователя, а соответствующее значение — хэш пароля (без имени пользователя в формате htpasswd). Например:

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

Примечание: в настоящее время поддерживаются только хэши паролей в формате htpasswd, сгенерированные с использованием алгоритма apr1.

CR

В ALB CR добавлены конфигурационные параметры, связанные с auth, которые можно настроить в ALB/Frontend/Rule CR. Во время выполнения ALB преобразует аннотации в Ingress в правила.

```yaml
auth:
  # Конфигурация базовой аутентификации
  basic:
    #  string; соответствует nginx.ingress.kubernetes.io/auth-type: basic
    auth_type: "basic"
    #  string; соответствует nginx.ingress.kubernetes.io/auth-realm
    realm: "Restricted Access"
    #  string; соответствует nginx.ingress.kubernetes.io/auth-secret
    secret: "ns/name"
    #  string; соответствует nginx.ingress.kubernetes.io/auth-secret-type
    secret_type: "auth-map|auth-file"
  # Конфигурация Forward аутентификации
  forward:
    #  boolean; соответствует nginx.ingress.kubernetes.io/auth-always-set-cookie
    always_set_cookie: true
    #  string; соответствует nginx.ingress.kubernetes.io/auth-proxy-set-headers
    auth_headers_cm_ref: "ns/name"
    #  string; соответствует nginx.ingress.kubernetes.io/auth-request-redirect
    auth_request_redirect: "/login"
    #  string; соответствует nginx.ingress.kubernetes.io/auth-method
    method: "GET"
    #  string; соответствует nginx.ingress.kubernetes.io/auth-signin
    signin: "/signin"
    #  string; соответствует nginx.ingress.kubernetes.io/auth-signin-redirect-param
    signin_redirect_param: "redirect_to"
    #  []string; соответствует nginx.ingress.kubernetes.io/auth-response-headers
    upstream_headers:
      - "X-User-ID"
      - "X-User-Name"
      - "X-User-Email"
    #  string; соответствует nginx.ingress.kubernetes.io/auth-url
    url: "http://auth-service/validate"
```

Auth поддерживает конфигурацию в:

- `.spec.config.auth` ALB CR
- `.spec.config.auth` Frontend CR
- `.spec.config.auth` Rule CR

Порядок наследования: Alb > Frontend > Rule. Если дочерний CR не настроен, используется конфигурация родительского CR.

Специальная аннотация Ingress для ALB

При обработке Ingress ALB определяет приоритет на основе префикса аннотации. Приоритет от высокого к низкому:

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

Это позволяет решать проблему совместимости с ingress-nginx и задавать конфигурацию auth на конкретном пути Ingress.

Auth-Enable

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

Новая аннотация, добавленная ALB, используется для указания, включена ли функция аутентификации для данного Ingress.

Другие функции, связанные с Auth в Ingress-Nginx

Global-Auth

В ingress-nginx можно задать глобальный auth через ConfigMap. Это эквивалентно настройке auth для всех Ingress. В ALB auth можно настроить в ALB2 и FT CR. Правила под ними наследуют эти настройки.

No-Auth-Locations

В ALB можно отключить функцию auth для данного Ingress, настроив аннотацию: `alb.ingress.cpaas.io/auth-enable: "false"` в Ingress.

Примечание: несовместимые моменты с Ingress-Nginx

1. Не поддерживается auth-keepalive
2. Не поддерживается auth-snippet
3. Не поддерживается auth-cache
4. Не поддерживается auth-tls
5. Basic-auth поддерживает только basic, digest не поддерживается
6. Basic-auth basic поддерживает только алгоритм apr1, bcrypt, sha256 и др. не поддерживаются

Устранение неполадок

1. Проверьте логи контейнера Nginx в pod ALB
2. Проверьте заголовок `X-ALB-ERR-REASON` в ответе

Развертывание высокодоступного VIP для ALB

Для обеспечения высокой доступности **ALB** требуется VIP. Существует два способа получения VIP.

Содержание

Метод 1: Использование Service типа LoadBalancer для предоставления VIP

При создании ALB в режиме сети `container` система автоматически создает Service типа LoadBalancer, который предоставляет VIP для этого ALB.

Перед использованием убедитесь, что кластер поддерживает Service типа LoadBalancer. Вы можете использовать встроенную реализацию платформы. Для настройки смотрите Configure MetalLB.

После подготовки MetalLB вы можете добавить следующие аннотации в `alb.spec.config.vip.lbSvcAnnotations` для настройки поведения MetalLB. Смотрите ALB networking configuration.

Аннотация	Описание
`metallb.universe.tf/loadBalancerIPs`	VIP, выделяемые для Service, через запятую. Используйте несколько значений для мульти-VIP или dual-stack, например: `192.0.2.10,2001:db8::10`.
`metallb.universe.tf/address-pool`	Пул адресов MetalLB, из которого выделяются IP.

Метод 2: Использование внешнего ALB устройства для предоставления VIP

- Перед развертыванием уточните у сетевого инженера IP-адрес (публичный IP, приватный IP, VIP) или доменное имя сервиса ALB. Если вы хотите использовать доменное имя в качестве адреса для внешнего трафика к ALB, необходимо заранее получить доменное имя и настроить его разрешение. Рекомендуется использовать коммерческое устройство Load Balancer для предоставления VIP, в противном случае можно использовать Pure Software Data Center LB Solution (Alpha).
- В зависимости от бизнес-сценария внешний ALB должен быть настроен на проверку работоспособности (health check) всех используемых портов, чтобы снизить время простоя при обновлении ALB. Конфигурация health check следующая:

Параметры проверки работоспособности	Описание
Порт	• Для глобальных кластеров укажите: 11782. • Для бизнес-кластеров укажите: 1936.
Протокол	Тип протокола для проверки работоспособности, рекомендуется использовать TCP.

Параметры проверки работоспособности	Описание
Таймаут ответа	Время ожидания получения ответа проверки, рекомендуется настроить 2 секунды.
Интервал проверки	Интервал между проверками, рекомендуется настроить 5 секунд.
Порог неработоспособности	Количество последовательных неудач, после которых статус backend-сервера считается неработоспособным, рекомендуется настроить 3 раза.

Модификация заголовков

Содержание

Основные понятия

При получении запроса модификация заголовков позволяет изменить заголовки запроса перед его пересылкой на backend. Аналогично, при получении ответа она позволяет изменить заголовки ответа перед их возвратом клиенту.

Использование аннотаций

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`

Значения аннотаций — это JSON-строки, содержащие конфигурацию.

```
type RewriteRequestConfig struct {
    Headers       map[string]string   `json:"headers,omitempty"`         // установить заголовок
    HeadersVar    map[string]string   `json:"headers_var,omitempty"`     // установить заголовок, значения которого - имена переменных
    HeadersRemove []string            `json:"headers_remove,omitempty"`  // удалить заголовок
    HeadersAdd    map[string][]string `json:"headers_add,omitempty"`     // добавить заголовок, значения которого могут быть множественными
    HeadersAddVar map[string][]string `json:"headers_add_var,omitempty"` // добавить заголовок, значения которого могут быть множественными и являются именами переменных
}
type RewriteResponseConfig struct {
    Headers       map[string]string   `json:"headers,omitempty"`         // установить заголовок
    HeadersRemove []string            `json:"headers_remove,omitempty"`  // удалить заголовок
    HeadersAdd    map[string][]string `json:"headers_add,omitempty"`     // добавить заголовок, значения которого могут быть множественными
}
```

Примечание: в картах `*_var` ключ — имя заголовка, а значение — имя переменной контекста ALB. Например, добавьте следующую аннотацию в Ingress:

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

добавит ключ `x-my-host` со значением заголовка host запроса в заголовки запроса. Вы можете обратиться к nginx variable ↗ для имен переменных.

ALB предоставляет дополнительные переменные:

имя переменной	описание
`first_forward_or_remote_addr`	первый адрес из заголовка Forwarded или удалённый адрес, по умолчанию `remote_addr`
`first_forward`	первый адрес из заголовка Forwarded, по умолчанию пустая строка

Примеры

Чтобы добавить заголовок Authorization из cookie, можно использовать:

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

Чтобы установить HSTS, можно использовать:

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

HTTP Redirect

Содержание

Основные понятия

HTTP redirect — это функция, предоставляемая ALB. Она напрямую возвращает HTTP-код 30x для запроса, который соответствует правилу. Заголовок Location используется для указания клиенту перенаправиться на новый URL.

ALB поддерживает настройку редиректа на уровне порта и правила.

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
```

Редирект может быть настроен на:

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

Аннотация Ingress

Аннотация	Описание
nginx.ingress.kubernetes.io/permanent-redirect	Соответствует URL в CR, по умолчанию устанавливает код 301
nginx.ingress.kubernetes.io/permanent-redirect-code	Соответствует полю code в CR
nginx.ingress.kubernetes.io/temporal-redirect	Соответствует URL в CR, по умолчанию устанавливает код 302

Аннотация	Описание
nginx.ingress.kubernetes.io/temporal-redirect-code	Соответствует полю code в CR
nginx.ingress.kubernetes.io/ssl-redirect	Соответствует scheme в CR, по умолчанию устанавливает scheme HTTPS
nginx.ingress.kubernetes.io/force-ssl-redirect	Соответствует scheme в CR, по умолчанию устанавливает scheme HTTPS

SSL-Redirect

1. SSL-redirect и force-ssl-redirect отличаются тем, что SSL-redirect действует только если у ingress есть сертификат для соответствующего домена, тогда как force-ssl-redirect действует независимо от наличия сертификата.
2. Для HTTPS-портов, если настроен только SSL-redirect, редирект не будет установлен.

Редирект на уровне порта

Когда редирект настроен на уровне порта, *все запросы* к этому порту будут перенаправлены согласно конфигурации редиректа.

Редирект на уровне правила

Когда редирект настроен на уровне правила, запросы, соответствующие этому правилу, будут перенаправлены согласно конфигурации редиректа.

L4/L7 Таймаут

Содержание

Основные понятия

L4/L7 таймаут — это функция, предоставляемая ALB. Она используется для настройки времени ожидания таймаута для L4/L7 прокси.

Таймаут реализован с помощью Lua-скрипта, и Nginx **не нужно перезагружать** при его изменении.

CRD

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

Конфигурация может быть задана на:

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

Что означает таймаут

Существует три типа таймаутов:

1. **proxy_connect_timeout_ms**: Определяет время ожидания установления соединения с upstream-сервером. Если соединение не может быть установлено в течение этого времени, запрос завершается с ошибкой.
2. **proxy_read_timeout_ms**: Определяет время ожидания чтения ответа от upstream-сервера. Таймаут устанавливается между двумя последовательными операциями чтения, а не на весь ответ целиком. Если в течение этого времени не поступают данные, соединение закрывается.
3. **proxy_send_timeout_ms**: Определяет время ожидания отправки запроса upstream-серверу. Аналогично таймауту чтения, он устанавливается между двумя последовательными операциями записи. Если в течение этого времени данные не могут быть отправлены, соединение закрывается.

Аннотации Ingress

Аннотация	Описание
nginx.ingress.kubernetes.io/proxy-connect-timeout	Соответствует proxy_connect_timeout_ms в CRD
nginx.ingress.kubernetes.io/proxy-read-timeout	Соответствует proxy_read_timeout_ms в CRD
nginx.ingress.kubernetes.io/proxy-send-timeout	Соответствует proxy_send_timeout_ms в CRD

Таймаут на уровне порта

Вы можете настроить таймаут непосредственно на порту, который используется как таймаут L4.

ModSecurity

ModSecurity — это открытый Web Application Firewall (WAF), предназначенный для защиты веб-приложений от вредоносных атак. Он поддерживается сообществом с открытым исходным кодом и поддерживает различные языки программирования и веб-серверы. Платформенный Load Balancer (ALB) поддерживает настройку ModSecurity, позволяя создавать индивидуальные конфигурации на уровне Ingress.

Содержание

Терминология

Термин	Объяснение
owasp-core-rules	OWASP Core Rule Set — это набор правил с открытым исходным кодом, используемый для обнаружения и предотвращения распространённых атак на веб-приложения.

Процедура эксплуатации

Настройте ModSecurity, добавив аннотации в YAML-файл соответствующего ресурса или настроив CR.

Способ первый: Добавление аннотаций

Добавьте следующие аннотации в поле metadata.annotations соответствующего YAML-файла для настройки ModSecurity.

- **Аннотации, совместимые с Ingress-Nginx**

Аннотация	Тип	Применимый объект	Объясне
nginx.ingress.kubernetes.io/enable-modsecurity	bool	Ingress	Включить ModSecu
nginx.ingress.kubernetes.io/enable-owasp-core-rules	bool	Ingress	Включить OWASP C Rule Set.
nginx.ingress.kubernetes.io/modsecurity-transaction-id	string	Ingress	Использу для идентиф уникальн транзакц каждого запроса, помогает логирова отладке.
nginx.ingress.kubernetes.io/modsecurity-snippet	string	Ingress, ALB, FT, Rule	Позволяе пользова вставлят собствен

Аннотация	Тип	Применимый объект	Объясне
			конфигур ModSecu для удовлетв специфи требован безопасн

- **Специальные аннотации ALB**

Аннотация	Тип	Применимый объект	Объяснение
alb.modsecurity.cpaas.io/use-recommend	bool	Ingress	Включить или отключить рекомендуемые правила ModSecurity; установите `true` для применения предопределённого набора правил безопасности.
alb.modsecurity.cpaas.io/cmref	string	Ingress	Ссылка на конкретные конфигурации, например, можно загрузить пользовательские настройки безопасности, указав путь ссылки на ConfigMap (`$ns/$name#$section`)

Способ второй: Настройка CR

1. Откройте файл конфигурации ALB, FT или Rule, который необходимо настроить.

2. Добавьте следующие поля в spec.config по необходимости.

```
{ "modsecurity": {
      "enable": true, # Включить или отключить ModSecurity
      "transactionId": "$xx", # Использовать ID из Nginx
      "useCoreRules": true, # Добавить modsecurity_rules_file /etc/nginx/owasp-
modsecurity-crs/nginx-modsecurity.conf
      "useRecommend": true, # Добавить modsecurity_rules_file
/etc/nginx/modsecurity/modsecurity.conf
      "cmRef": "$ns/$name#$section", # Добавить конфигурацию из ConfigMap
    } }
```

3. Сохраните и примените файл конфигурации.

Связанные пояснения

Переопределение

Если ModSecurity не настроен в Rule, он попытается найти конфигурацию в FT; если в FT конфигурация отсутствует, будет использована конфигурация из ALB.

Пример конфигурации

В следующем примере развёртывается ALB с именем `waf-alb` и демонстрационное backend-приложение с именем `hello`. Кроме того, развёртывается Ingress с именем `ing-waf-enable`, который определяет маршрут `/waf-enable` и настраивает правила ModSecurity. Любой запрос, содержащий параметр запроса `test`, значение которого включает строку `test`, будет заблокирован.

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

```
  rules:
    - http:
        paths:
          - backend:
              service:
                name: hello
                port:
                  number: 80
            path: /waf-not-enable
            pathType: ImplementationSpecific
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello
spec:
  replicas: 1
  selector:
    matchLabels:
      service.cpaas.io/name: hello
      service_name: hello
  template:
    metadata:
      labels:
        service.cpaas.io/name: hello
        service_name: hello
    spec:
      containers:
        - name: hello-world
          image: 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

Содержание

Основные понятия

1. ALB поддерживает конфигурацию keepalive на уровне порта. Она может быть настроена на frontend.
2. Keepalive работает **между клиентом и ALB, а не между ALB и backend**.
3. Реализуется через конфигурацию Nginx, и Nginx **требует и автоматически перезагружается** при изменении конфигурации.
4. TCP keepalive и HTTP keepalive — это два разных понятия:

    1. **TCP keepalive** — это функция протокола TCP, которая периодически отправляет probe-пакеты для проверки, жива ли соединение при отсутствии передачи данных. Это помогает обнаруживать и очищать «мертвые» соединения.

    2. **HTTP keepalive** (также известный как persistent connections) позволяет нескольким HTTP-запросам использовать одно и то же TCP-соединение, избегая накладных расходов на установку новых соединений. Это улучшает производительность за счёт снижения задержек и использования ресурсов.

CRD

```
keepalive:
  properties:
    http:
      description: Downstream L7 keepalive
      properties:
        header_timeout:
          description: Keepalive header timeout. Default is not set.
          type: string
        requests:
          description: Keepalive requests. Default is 1000.
          type: integer
        timeout:
          description: Keepalive timeout. Default is 75s.
          type: string
      type: object
    tcp:
      description: TCPKeepAlive defines TCP keepalive parameters (SO_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
```

Её можно настроить только на Frontend в `.spec.config.keepalive`.

Использование OAuth Proxy с ALB

Содержание

Overview

В этом документе показано, как использовать OAuth Proxy с ALB для реализации внешней аутентификации.

Procedure

Выполните следующие шаги для использования данной функции:

1. Разверните kind

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

2. Разверните 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. Разверните тестовое приложение

- Создайте github oauth app ↗

> Обратите внимание, что `$GITHUB_CLIENT_ID` и `$GITHUB_CLIENT_SECRET` будут получены на этом шаге и должны быть установлены в переменные окружения

- Настройте DNS

> Здесь мы используем echo.com в качестве домена приложения, auth.alb.echo.com и alb.echo.com

- Разверните oauth-proxy

> oauth2-proxy должен иметь доступ к github, возможно потребуется установка переменной окружения HTTPS_PROXY

```
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. Настройте ingress

   Мы настроим два ingress: auth.alb.echo.com и 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

- После завершения операции будут развернуты alb, oauth-proxy и тестовое приложение.
- При обращении к alb.echo.com вы будете перенаправлены на страницу аутентификации github, после проверки вы увидите вывод приложения.

Настройка GatewayApi Gateway через ALB

Входящий шлюз (Gateway) — это экземпляр, развернутый из Gateway Class. Он создает слушатели для перехвата внешнего трафика на указанных доменных именах и портах. Вместе с правилами маршрутизации он может направлять указанный внешний трафик на соответствующие бэкенд-экземпляры.

Создайте входящий шлюз для более детального распределения сетевых ресурсов.

Содержание

Терминология

Название ресурса	Обзор	Инструкция по использованию
Gateway Class	В стандартной документации Gateway API Gateway Class определяется как шаблон для создания шлюзов. Различные шаблоны могут создавать входящие шлюзы для разных бизнес-сценариев, что облегчает быстрое управление трафиком.	Платформа включает выделенные Gateway Classes.
Inbound Gateway	Входящий шлюз соответствует конкретным экземплярам ресурсов, и пользователь может эксклюзивно использовать все ресурсы прослушивания и вычислительные ресурсы этого входящего шлюза. Это конфигурация правил маршрутизации, действующая для слушателя. При обнаружении внешнего трафика шлюз распределяет его на бэкенд-экземпляры согласно правилам маршрутизации.	Его можно рассматривать как экземпляр балансировщика нагрузки.
Route Rule	Правила маршрутизации определяют набор правил для распределения трафика от шлюза к сервисам. В настоящее время стандартно поддерживаемые типы правил маршрутизации в Gateway API включают HTTPRoute, TCPRoute, UDPRoute и др.	Платформа в настоящее время поддерживает прослушивание протоколов HTTP, HTTPS, TCP и UDP.

Предварительные требования

Администратор платформы должен убедиться, что кластер поддерживает внутреннюю маршрутизацию типа LoadBalancer. Для публичных облачных кластеров должен быть установлен LoadBalancer Service Controller. В непубличных облаках платформа предоставляет функцию пула внешних адресов, которая позволяет внутренней маршрутизации типа LoadBalancer автоматически получать IP из пула внешних адресов для внешнего доступа после завершения настройки.

Пример Gateway и пользовательского ресурса Alb2 (CR)

```yaml
# 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. Какой ALB использует этот шлюз.
2. См. описание Gateway Class ниже.
3. Формат имени `ALB2` : `{gatewayName}-{random}` .
4. Имя шлюза.

Создание Gateway через веб-консоль

1. Перейдите в **Container Platform**.

2. В левой навигационной панели выберите **Network** > **Inbound Gateway**.

3. Нажмите **Create Inbound Gateway**.

4. Следуйте инструкциям для настройки конкретных параметров.

Параметр	Описание
Name	Имя входящего шлюза.
Gateway Class	Gateway Class определяет поведение шлюза, аналогично концепции классов хранения (StorageClasses); это ресурс кластера. **Dedicated**: Входящий шлюз будет соответствовать конкретному экземпляру ресурса, и пользователь сможет использовать все слушатели и вычислительные ресурсы этого шлюза.
Specification	Можно выбрать рекомендуемый сценарий использования или настроить лимиты ресурсов по своему усмотрению.
Access Address	Адрес входящего шлюза, который по умолчанию получается автоматически.
Internal Routing Annotation	Используется для объявления конфигурации или возможностей внутренней маршрутизации типа LoadBalancer. Для подробной информации об аннотациях

Параметр	Описание
	смотрите LoadBalancer type internal routing annotation instructions.

5. Нажмите **Create**.

Создание Gateway через CLI

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

Просмотр ресурсов, созданных платформой

После создания входящего шлюза платформа автоматически создает множество ресурсов. Не удаляйте перечисленные ниже ресурсы.

Ресурсы, создаваемые по умолчанию	Имя
Ресурс типа ALB2	*name-lb-random*
Deployment	*name-lb-random*
Внутренняя маршрутизация	• *name-lb-random* • *name-lb-random-lb-random*
Конфигурационный словарь	• *name-lb-random-port-info* • *name-lb-random*
Service Account	*name-lb-random-serviceaccount*

Обновление шлюзов

NOTE

Обновление входящего шлюза приведет к прерыванию сервиса на 3-5 минут. Пожалуйста, выбирайте подходящее время для этой операции.

Обновление Gateway через веб-консоль

1. Зайдите в **Container Platform**.
2. В левой навигационной панели выберите **Network** > **Inbound Gateway**.
3. Нажмите ⋮ > **Update**.
4. Обновите конфигурацию входящего шлюза по необходимости.

   Примечание: Пожалуйста, разумно задавайте спецификации в соответствии с требованиями бизнеса.
5. Нажмите **Update**.

Добавление слушателя

Отслеживает трафик по указанным доменным именам и перенаправляет его на бэкенд-экземпляры согласно привязанным правилам маршрутизации.

Предварительные требования

- Если необходимо отслеживать протокол HTTP, заранее свяжитесь с администратором для подготовки **доменного имени**.

- Если необходимо отслеживать протокол HTTPS, заранее свяжитесь с администратором для подготовки **доменного имени** и **сертификата**.

Добавление слушателя через веб-консоль

1. В левой навигационной панели выберите **Network** > **Inbound Gateway**.
2. Нажмите на ***Имя входящего шлюза***.
3. Нажмите **Add Listener**.
4. Следуйте инструкциям для настройки конкретных параметров.

Параметр	Описание
Протокол и порт слушателя	В настоящее время поддерживается мониторинг протоколов HTTP, HTTPS, TCP и UDP, можно ввести порт для мониторинга, например: `80`. **Примечание**: • При совпадении портов типы слушателей HTTP, HTTPS и TCP не могут сосуществовать; можно выбрать только один протокол. • При использовании HTTP или HTTPS и совпадении портов доменные имена должны отличаться.
Доменное имя	Выберите доступное доменное имя в текущем пространстве имен, используемое для мониторинга сетевого трафика, обращающегося к этому домену. **Подсказка**: протоколы TCP и UDP не поддерживают выбор доменных имен.

5. Нажмите **Create**.

Добавление слушателя через 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"
        }
      ]
    }
  }'
```

Создание правил маршрутизации

Правила маршрутизации предоставляют политики маршрутизации для входящего трафика, аналогично входящим правилам (Kubernetes Ingress). Они открывают сетевой трафик, отслеживаемый шлюзом, для внутренней маршрутизации кластера (Kubernetes Service), облегчая стратегии маршрутизации и пересылки. Ключевое отличие в том, что они нацелены на разные объекты сервисов: входящие правила обслуживают Ingress Controller, а правила маршрутизации — Ingress Gateway.

После настройки прослушивания в ingress-шлюзе шлюз будет в реальном времени отслеживать трафик с указанных доменов и портов. Правила маршрутизации могут перенаправлять входящий трафик на бэкенд-экземпляры по желанию.

Пример пользовательского ресурса HTTPRoute (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) Доступные типы: `HTTPRoute`, `TCPRoute`, `UDPRoute`.

(2) Имя слушателя `Gateway`.

NOTE

Если для объекта **Path** в правиле маршрутизации типа HTTPRoute нет совпадающего правила, автоматически добавляется правило с режимом PathPrefix и значением /.

Создание маршрута через веб-консоль

1. Зайдите в **Container Platform**.

2. В левой навигационной панели выберите **Network** > **Route Rules**.

3. Нажмите **Create Route Rule**.

4. Следуйте инструкциям для настройки параметров.

Параметр	Описание
Тип маршрута	В настоящее время поддерживаются типы маршрутов: HTTPRoute, TCPRoute, UDPRoute. **Совет:** HTTPRoute поддерживает публикацию на слушатели протоколов HTTP и HTTPS.
Публикация на слушатель	В левой панели выберите созданный **Ingress Gateway**, в правой — созданный **Listener**. Платформа опубликует созданные правила маршрутизации на выбранный слушатель, позволяя шлюзу перенаправлять перехваченный трафик на указанные бэкенд-экземпляры. **Примечание:** Нельзя публиковать правила маршрутизации на слушатель с портом **11782** или уже подключенными маршрутами TCP или UDP.
Совпадение	Можно добавить одно или несколько правил совпадения для перехвата трафика, соответствующего требованиям, например, **перехват трафика с указанным Path**, **перехват трафика с указанным методом** и т.д. **Примечание:** • Нажмите **Add**; при добавлении нескольких правил маршрутизации связь между ними — 'AND', все правила должны совпадать для срабатывания.

Параметр	Описание
	• Нажмите **Add Match**; при добавлении нескольких групп правил связь между группами — 'OR', срабатывает любая группа. • TCPRoute и UDPRoute не поддерживают настройку правил совпадения. • При совпадении по объекту **path** и методах **Exact** или **PathPrefix** значение **value** должно начинаться с "/" и не должно содержать символы "//", "/./", "/../", "%2f", "%2F", "#", "/..", "/." и т.п.
Действие	Можно добавить одно или несколько действий для обработки перехваченного трафика. • Header: Заголовок HTTP-сообщения содержит метаданные, предоставляющие дополнительную информацию о запросе или ответе. Изменяя поля заголовков, сервер может влиять на обработку запроса и ответа. • Redirect: Совпадающий URL будет обработан указанным образом, после чего запрос будет инициирован заново. • Rewrite: Совпадающий URL будет обработан указанным образом, после чего запрос будет перенаправлен на другой путь ресурса или имя файла. **Примечание:** • Нажмите **Add**; при добавлении нескольких правил действий платформа выполнит все действия по порядку согласно отображаемой последовательности. • TCPRoute и UDPRoute не поддерживают настройку правил действий. • В одном правиле маршрутизации не может быть несколько действий типа **Header** с одинаковым **value**.

Параметр	Описание
	• В одном правиле маршрутизации может быть только одно действие типа **Redirect** или **Rewrite**, и только один режим из **FullPath** или **PrefixPath**. • Для использования операции **PrefixPath** сначала добавьте правило совпадения с режимом **PathPrefix**.
Бэкенд-экземпляр	После вступления правила в силу трафик будет перенаправлен на бэкенд-экземпляр согласно выбранным внутренним маршрутам и портам в текущем пространстве имен. Можно также задать веса, при этом экземпляры с большим весом будут выбираться с большей вероятностью. **Совет:** Процент рядом с весом показывает вероятность передачи трафика на этот экземпляр, рассчитываемую как отношение текущего веса к сумме всех весов.

5. Нажмите **Create**.

Создание маршрута через CLI

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

Привязка NIC в ALB

По умолчанию ALB слушает на `0.0.0.0` для ipv4 и `::` для ipv6. В некоторых сценариях безопасности требуется привязать его к конкретной сетевой карте (NIC).

Содержание

Для встроенного ALB в кластере

По умолчанию встроенный ALB разворачивается в каждом кластере. В глобальном кластере он должен называться 'global-alb2', а в остальных кластерах — 'cpaas-system'.

Замените `$CLUSTER` и `$NIC` на фактические значения кластера и NIC. Если вы используете Alive (Alauda Container Platform Virtual IP Management), необходимо добавить `alive` в список nic.

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

По умолчанию ALB включает IPv6 в кластере с одной стековой конфигурацией. Однако при использовании bindnic указанная NIC может не иметь IPv6-адреса. В таких случаях ALB всё равно попытается привязаться к `::*`. В качестве обходного решения можно отключить IPv6.

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

Для пользовательского ALB

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

Принятие решений по выбору производительности ALB

Для предлагаемых платформой спецификаций для **малых**, **средних**, **больших** и **кастомных** производственных сред, а также методов выделения ресурсов для **инстансов** и **портов**, можно ориентироваться на следующие рекомендации при развертывании.

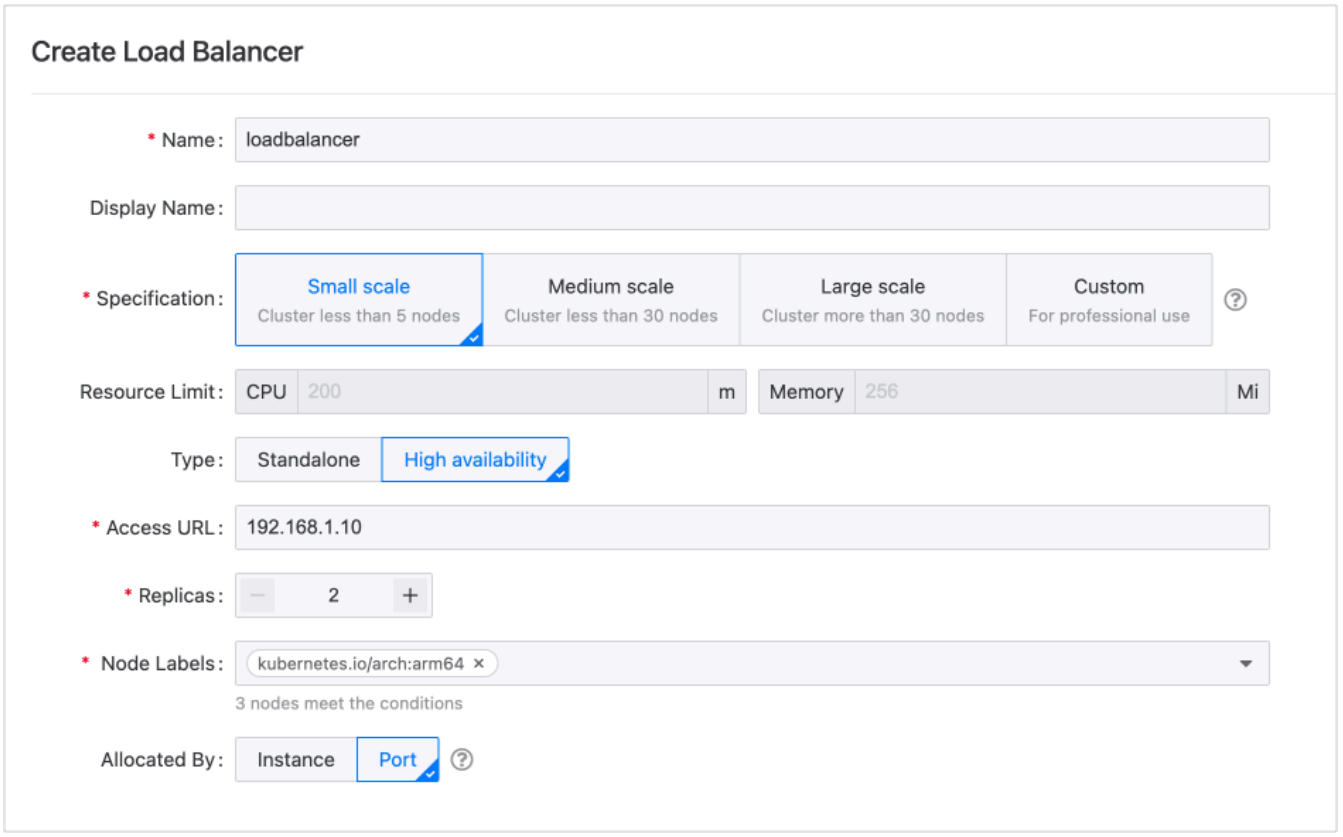
Содержание

Малое производственное окружение

Для бизнесов меньшего масштаба, например, при наличии не более 5 узлов в кластере и использовании только для запуска стандартных приложений, достаточно **одного** балансировщика нагрузки. Рекомендуется использовать его в режиме **высокой доступности** с минимум 2 репликами для обеспечения стабильности среды.

Можно изолировать балансировщик нагрузки с помощью **изоляции по портам**, что позволит нескольким проектам совместно его использовать.

Пиковый QPS, измеренный в лабораторных условиях для этой спецификации, составляет примерно 300 запросов в секунду.

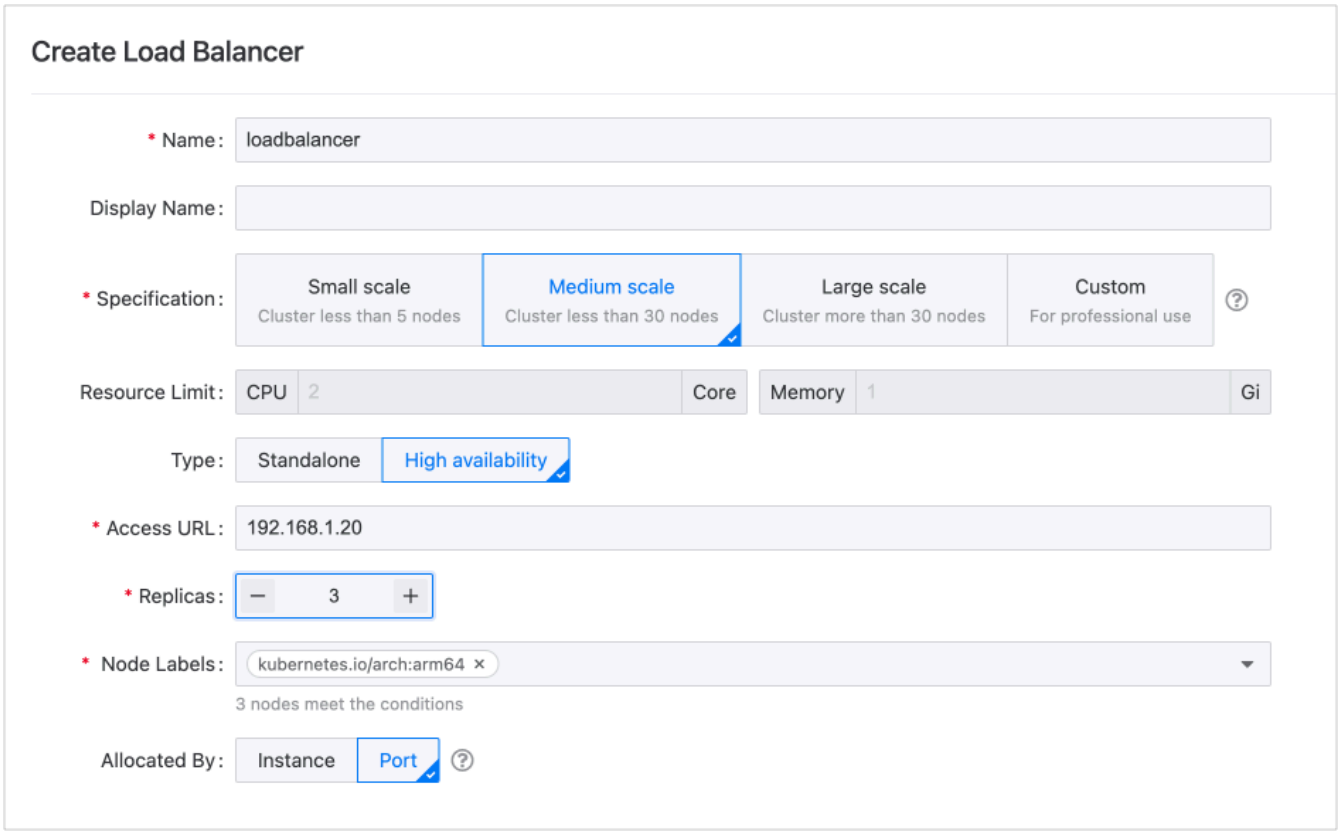


Среднее производственное окружение

Когда объем бизнеса достигает определенного масштаба, например, при наличии не более 30 узлов в кластере и необходимости обработки высококонкурентных бизнес-задач наряду с запуском стандартных приложений, **одного** балансировщика нагрузки по-прежнему будет достаточно. Рекомендуется использовать режим **высокой доступности** с минимум 3 репликами для поддержания стабильности среды.

Можно использовать как метод **изоляции по портам**, так и метод выделения **инстансов** для совместного использования балансировщика нагрузки несколькими проектами. Конечно, также можно создавать новые балансировщики нагрузки для выделенного использования ключевыми проектами.

Пиковый QPS, измеренный в лабораторных условиях для этой спецификации, составляет около 10 000 запросов в секунду.

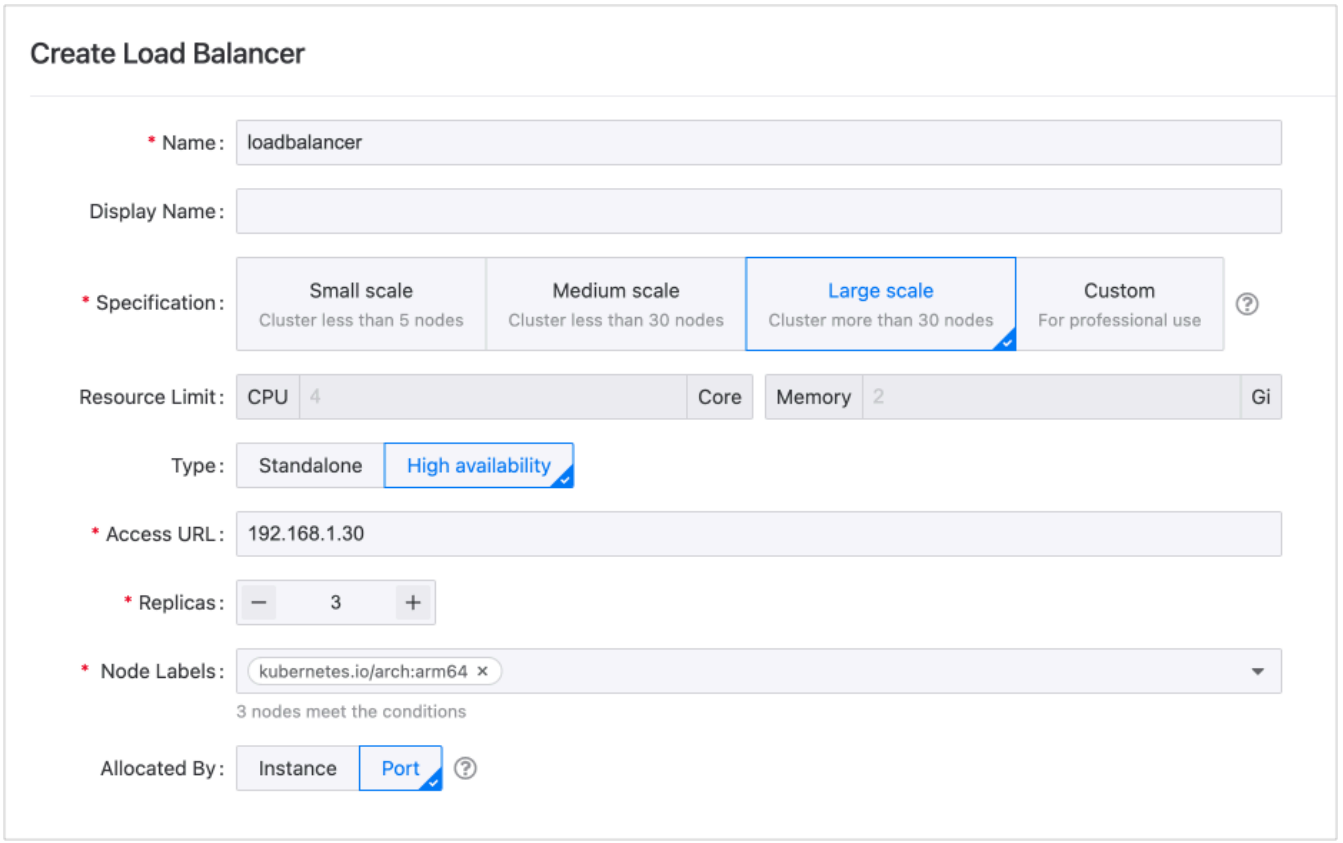


Большое производственное окружение

Для больших объемов бизнеса, например, при наличии более 30 узлов в кластере и необходимости обработки высококонкурентных бизнес-задач, а также долгоживущих соединений с данными, рекомендуется использовать **несколько** балансировщиков нагрузки, каждый из которых работает в режиме **высокой доступности** с минимум 3 репликами для обеспечения стабильности среды.

Можно изолировать балансировщик нагрузки как с помощью **изоляции по портам**, так и с помощью выделения **инстансов** для совместного использования несколькими проектами. Также можно создавать новые балансировщики нагрузки для эксклюзивного использования ключевыми проектами.

Пиковый QPS, измеренный в лабораторных условиях для этой спецификации, составляет примерно 20 000 запросов в секунду.



Рекомендации по развертыванию в особых сценариях

Сценарий	Рекомендации по развертыванию
Функциональное тестирование	Рекомендуется развернуть **один инстанс** балансировщика нагрузки.
Тестовая среда	Если тестовая среда соответствует определениям **малой** или **средней** среды, приведенным выше, достаточно использовать балансировщик нагрузки с **одной точкой**. Инстанс балансировщика нагрузки может быть **совместно использован несколькими проектами**.
Ключевые приложения	Рекомендуется использовать отдельные балансировщики нагрузки исключительно для ключевых приложений.
Передача больших объемов данных	Из-за минимального потребления памяти самим балансировщиком нагрузки достаточно резервировать 2Gi

Сценарий	Рекомендации по развертыванию
	памяти даже для спецификации **большого** размера. Однако, если бизнес требует передачи больших объемов данных, что приведет к значительному потреблению памяти, объем выделяемой памяти для балансировщика нагрузки следует увеличить соответственно. Рекомендуется постепенно расширять память балансировщика нагрузки в сценариях со спецификацией **кастомного** типа, внимательно отслеживая использование памяти, чтобы в итоге определить приемлемый размер памяти для разумных показателей использования.

Выбор режима использования балансировщика нагрузки

Режим использования	Преимущества	Недостатки
(Рекомендуется) Выделение балансировщика нагрузки как ресурса инстанса одному проекту	• Управление относительно простое. • Каждый проект имеет собственный балансировщик нагрузки, что обеспечивает изоляцию правил и разделение ресурсов без взаимных помех.	В режиме host network кластер должен иметь значительное количество доступных узлов для балансировщика нагрузки, что приводит к высоким требованиям к потреблению ресурсов.

Режим использования	Преимущества	Недостатки
Выделение балансировщика нагрузки как ресурса инстанса нескольким проектам	Управление относительно простое.	Поскольку все назначенные проекты имеют полные права на инстанс балансировщика нагрузки, при конфигурировании портов и правил одним проектом могут возникнуть следующие ситуации: • Правила, настроенные этим проектом, могут повлиять на другие проекты. • Ошибочные действия при настройке балансировщика нагрузки могут изменить настройки других проектов. • Трафик от конкретного бизнеса может повлиять на общую доступность инстанса балансировщика нагрузки.
Динамическое выделение ресурсов балансировщика нагрузки по портам, с использованием разных портов разными проектами	Правила между проектами изолируют их, обеспечивая отсутствие взаимных помех.	• Управление усложняется. Администраторам платформы необходимо активно планировать и выделять порты для проектов, а также настраивать внешние сервисные отображения.

Режим использования	Преимущества	Недостатки
		• Зрелость выделения по портам ниже. В настоящее время используется меньшим числом клиентов и требует дальнейшего совершенствования функций. • Конфликты ресурсов. Все сервисы, использующие один балансировщик нагрузки, могут столкнуться с ситуациями, когда один сервис негативно влияет на весь балансировщик.

Развертывание ALB

Содержание

ALB

ALB — это кастомный ресурс, представляющий балансировщик нагрузки. alb-operator, который по умолчанию встроен во все кластеры, отслеживает операции создания/обновления/удаления ресурсов ALB и в ответ создает соответствующие Deployment и Service.

Для каждого ALB создаётся соответствующий Deployment, который отслеживает все Frontend и Rule, прикрепленные к этому ALB, и маршрутизирует запросы к backend на основе этих конфигураций.

Предварительные требования

Высокая доступность **Load Balancer** требует VIP. Пожалуйста, обратитесь к разделу Настройка VIP.

Конфигурация ALB

Конфигурация ALB состоит из трёх частей.

```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
```

Конфигурация ресурсов

Поля, связанные с ресурсами, описывают конфигурацию развертывания alb.

Поле	Тип	Описание
`.spec.config.nodeSelector`	map[string]string	селектор узлов для alb
`.spec.config.replicas`	int, необязательно, по умолчанию 3	количество реплик alb
`.spec.config.resources.limits`	k8s container-resource, необязательно	лимиты контейнера nginx alb
`.spec.config.resources.requests`	k8s container-resource, необязательно	запросы контейнера nginx alb
`.spec.config.resources.alb.limits`	k8s container-resource, необязательно	лимиты контейнера alb
`.spec.config.resources.alb.requests`	k8s container-resource, необязательно	запросы контейнера alb
`.spec.config.antiAffinityKey`	string, необязательно, по умолчанию local	k8s antiAffinityKey

Конфигурация сети

Поля, связанные с сетью, описывают способ доступа к ALB. Например, в режиме `host` alb использует hostnetwork, и доступ к ALB осуществляется по IP узла.

Поле	Тип	Описание
`.spec.config.networkMode`	string: `host` или `container`, необязательно, по умолчанию `host`	В режиме `container` оператор создаёт Service типа LoadBalancer и использует его адрес как адрес ALB.

Поле	Тип	Описание
`.spec.address`	string, обязательно	можно вручную указать адрес alb
`.spec.config.vip.enableLbSvc`	bool, необязательно	Автоматически true в режиме `container`.
`.spec.config.vip.lbSvcAnnotations`	map[string]string, необязательно	Дополнительные аннотации для Service типа LoadBalancer.

Конфигурация проекта

Поле	Тип
`.spec.config.projects`	[]string, обязательно
`.spec.config.portProjects`	string, необязательно
`.spec.config.enablePortProject`	bool, необязательно

Добавление ALB в проект означает:

1. В веб-интерфейсе только пользователи данного проекта могут найти и настроить этот ALB.
2. Этот ALB будет обрабатывать ingress-ресурсы, принадлежащие этому проекту. Пожалуйста, обратитесь к ingress-sync.
3. В веб-интерфейсе правила, созданные в проекте X, не будут доступны для просмотра или настройки в проекте Y.

Если вы включаете порт-проект и назначаете диапазон портов проекту, это означает:

1. Вы не можете создавать порты, не входящие в назначенный проекту диапазон портов.

Настройка параметров

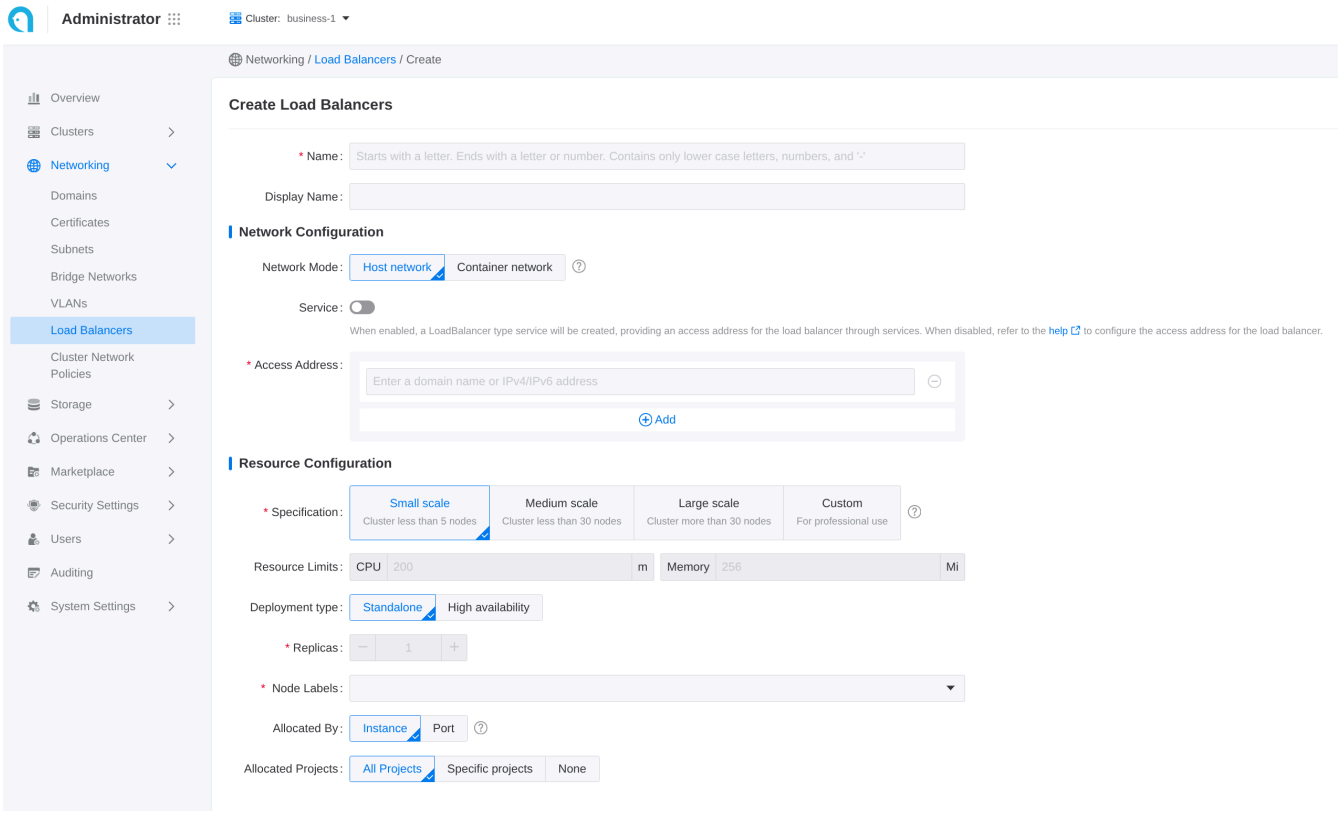
В alb CR есть некоторые глобальные настройки, которые можно изменить.

- bind-nic
- ingress-sync

Операции с ALB

Создание

Через веб-консоль



Некоторые общие настройки доступны в веб-интерфейсе. Следуйте этим шагам для создания балансировщика нагрузки:

1. Перейдите в раздел **Administrator**.
2. В левой боковой панели нажмите **Network Management** > **Load Balancer**.
3. Нажмите **Create Load Balancer**.

Каждое поле ввода в веб-интерфейсе соответствует полю CR:

Параметр	Описание
Assigned Address	`.spec.address`

Параметр	Описание
Allocated By	`Instance` означает режим проекта, где можно выбрать проект ниже, `port` — режим порт-проекта, где можно назначить диапазон портов после создания alb

Через CLI

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

Обновление

Через веб-консоль

> **NOTE**
>
> Обновление балансировщика нагрузки приведёт к прерыванию сервиса на 3–5 минут.
>
> Пожалуйста, выбирайте подходящее время для этой операции!

1. Войдите в **Administrator**.
2. В левой навигационной панели нажмите **Network Management** > **Load Balancer**.
3. Нажмите ⋮ > **Update**.
4. Обновите конфигурацию сети и ресурсов по необходимости.
   - Пожалуйста, устанавливайте спецификации разумно в соответствии с бизнес-требованиями. Также можно обратиться к соответствующему руководству Как правильно выделять CPU и память.
   - **Внутренний роутинг** поддерживает только обновление из состояния **Disabled** в **Enabled**.
5. Нажмите **Update**.

Удаление

Через веб-консоль

NOTE

После удаления балансировщика нагрузки связанные порты и правила также будут удалены и не подлежат восстановлению.

1. Войдите в **Administrator**.
2. В левой навигационной панели нажмите **Network Management** > **Load Balancer**.
3. Нажмите ⋮ > **Delete** и подтвердите.

Через CLI

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

Порты слушателя (Frontend)

Frontend — это кастомный ресурс, который определяет порт слушателя и протокол для ALB. Поддерживаемые протоколы: L7 (http|https|grpc|grpcs) и L4 (tcp|udp). В L4 Proxy frontend используется для прямой настройки backend-сервиса. В L7 Proxy frontend настраивает порты слушателя, а backend-сервис настраивается через rule. Если необходимо добавить HTTPS-порт слушателя, следует также обратиться к администратору для назначения TLS-сертификата текущему проекту для шифрования.

Предварительные требования

Сначала создайте ALB.

Конфигурация 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: Обязательно, указывает экземпляр ALB, к которому принадлежит этот `Frontend`.

(2) имя frontend: Формат `$alb_name-$port`.

(3) port: порт, на котором слушает.

(4) protocol: протокол, используемый на этом порту.

- L7 протоколы https|http|grpcs|grpc и L4 протоколы tcp|udp.
- При выборе HTTPS обязательно должен быть добавлен сертификат; для протокола gRPC добавление сертификата необязательно.
- При выборе протокола gRPC backendProtocol по умолчанию gRPC, который не поддерживает сохранение сессии. Если для gRPC установлен сертификат, балансировщик нагрузки снимет сертификат gRPC и перенаправит незашифрованный трафик gRPC на backend-сервис.
- В кластере Google GKE балансировщик нагрузки одного **типа контейнерной сети** не может одновременно иметь протоколы слушателя TCP и UDP.

5 certificate_name: для протоколов grpcs и https, указывает используемый сертификат по формату `$secret_ns/$secret_name`.

6 backendProtocol: протокол, используемый backend-сервисом.

7 По умолчанию `serviceGroup`:

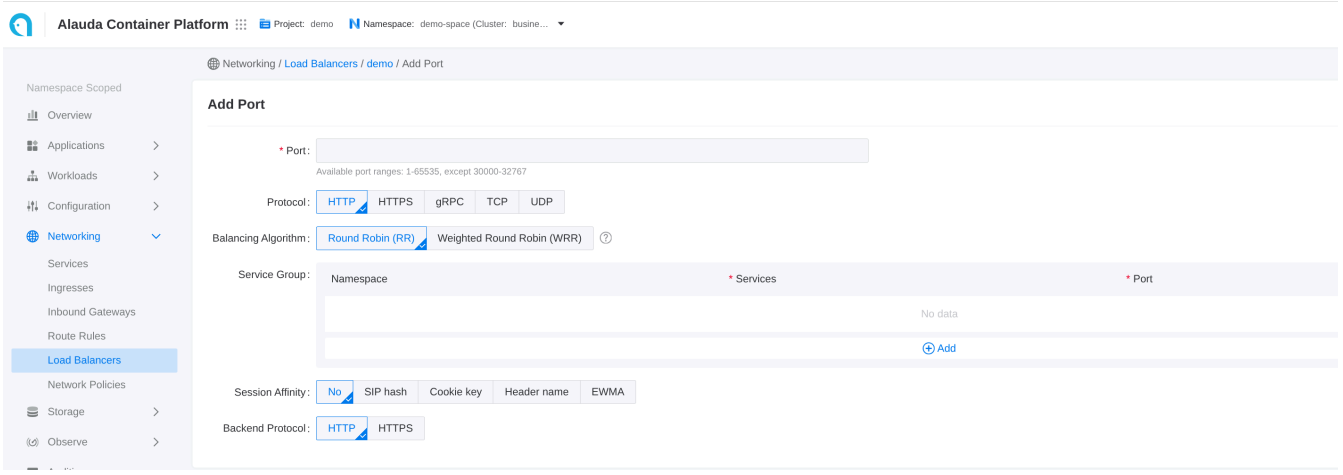
- L4 proxy: обязательно. ALB напрямую перенаправляет трафик в группу сервисов по умолчанию.
- L7 proxy: необязательно. ALB сначала сопоставляет правила на этом Frontend; если совпадений нет, используется группа сервисов по умолчанию.

8 session_affinity_policy

Операции с Frontend

Создание

Через веб-консоль



1. Перейдите в **Container Platform**.
2. В левой навигационной панели нажмите **Network** > **Load Balancing**.
3. Нажмите на имя балансировщика нагрузки, чтобы перейти на страницу деталей.
4. Нажмите **Add Port**.

Каждое поле ввода в веб-интерфейсе соответствует полю CR

Параметр	Описание
Session Affinity	`.spec.serviceGroup.session_affinity_policy`

Через CLI

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

Последующие действия

Для трафика с портов HTTP, gRPC и HTTPS, помимо группы внутреннего роутинга по умолчанию, можно задать более разнообразные правила сопоставления backend-сервисов rules. Балансировщик нагрузки сначала сопоставит соответствующий backend-сервис согласно заданным правилам; если совпадений не будет, он сопоставит backend-сервисы, соответствующие упомянутой группе внутреннего роутинга.

Связанные операции

Вы можете нажать на значок ⋮ справа на странице списка или нажать **Actions** в правом верхнем углу страницы деталей, чтобы при необходимости обновить маршрут по умолчанию или удалить порт слушателя.

> **NOTE**
>
> Если метод выделения ресурсов балансировщика нагрузки — **Port**, только администраторы могут удалять связанные порты слушателя в представлении **Administrator**.

Логи и мониторинг

Сочетая логи и данные мониторинга, вы можете быстро выявлять и устранять проблемы с балансировщиком нагрузки.

Просмотр логов

1. Перейдите в **Administrator**.
2. В левой навигационной панели нажмите **Network Management** > **Load Balancer**.
3. Нажмите на ***Имя Load Balancer***.
4. Во вкладке **Logs** просмотрите логи работы балансировщика нагрузки с точки зрения контейнера.

Метрики мониторинга

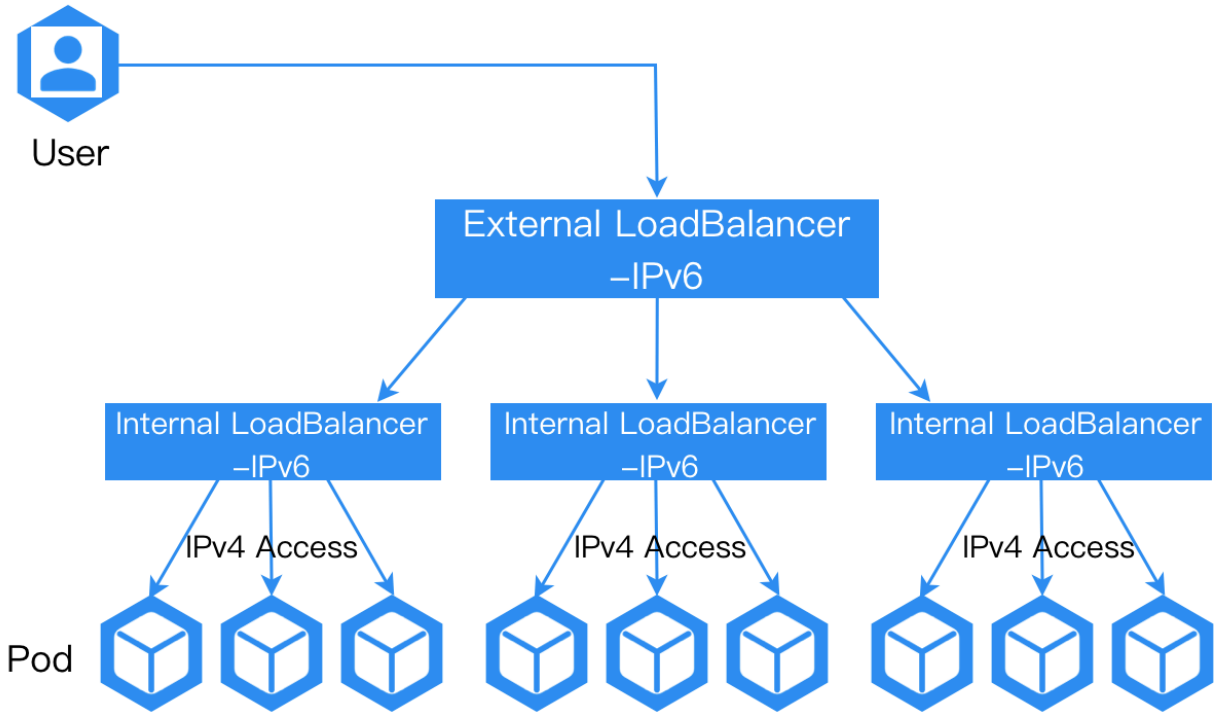
> **NOTE**
>
> В кластере, где расположен балансировщик нагрузки, должны быть развернуты сервисы мониторинга.

1. Перейдите в **Administrator**.
2. В левой навигационной панели нажмите **Network Management** > **Load Balancer**.
3. Нажмите на ***Имя Load Balancer***.
4. Во вкладке **Monitoring** просмотрите информацию о тенденциях метрик балансировщика нагрузки с точки зрения узла.
   - **Usage Rate**: текущая загрузка CPU и памяти балансировщика нагрузки на данном узле.
   - **Throughput**: общий входящий и исходящий трафик экземпляра балансировщика нагрузки.

Для более подробной информации о метриках мониторинга смотрите ALB Monitoring.

Проброс IPv6-трафика на IPv4-адреса внутри кластера через ALB

Настроив внешний балансировщик нагрузки для кластера, мы можем пробрасывать IPv6-трафик на внутренние IPv4-адреса внутри кластера. Это позволяет внедрить возможности IPv6 поверх существующей IPv4-сети, обеспечивая большую гибкость и масштабируемость архитектуры системы, а также лучше удовлетворяя разнообразные сетевые требования.



Содержание

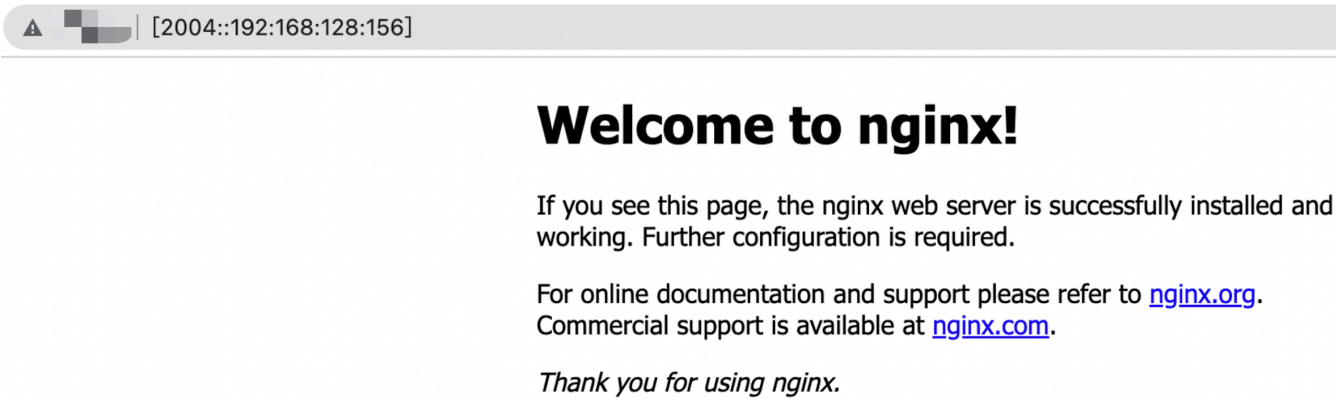
Метод настройки

1. Настройте IPv6-адрес для узла, на котором расположен балансировщик нагрузки.
2. Убедитесь, что у внешнего балансировщика нагрузки есть IPv6-адрес, и что трафик, обращающийся к IPv6-адресу балансировщика, может быть перенаправлен на IPv6-адрес узла, где расположен балансировщик.

После выполнения вышеуказанной настройки IPv4-сервисы, размещённые на балансировщике нагрузки, смогут предоставлять внешние возможности доступа по IPv6 через балансировщик.

Проверка результата

После настройки обращение к IPv6-адресу внешнего балансировщика нагрузки должно обеспечивать нормальный доступ к приложению.



OTel

OpenTelemetry (OTel) — это проект с открытым исходным кодом, направленный на предоставление независимого от поставщика стандарта для сбора, обработки и экспорта телеметрических данных в распределённых системах, таких как архитектуры микросервисов. Он помогает разработчикам проще анализировать производительность и поведение программного обеспечения, что облегчает диагностику и устранение проблем в приложениях.

Содержание

Терминология

Термин	Объяснение
Trace	Данные, отправляемые на OTel Server, представляющие собой набор связанных событий или операций, используемых для отслеживания потока запросов в распределённых системах; каждый Trace состоит из нескольких Spans.
Span	Независимая операция или событие внутри Trace, включающая время начала, длительность и другую релевантную информацию.
OTel Server	Сервер OTel, способный принимать и хранить данные Trace, например Jaeger, Prometheus и др.
Jaeger	Система распределённого трассирования с открытым исходным кодом, используемая для мониторинга и отладки архитектур микросервисов, поддерживающая интеграцию с OpenTelemetry.
Attributes	Пары ключ-значение, прикреплённые к Trace или Span для предоставления дополнительной контекстной информации. Включает Resource Attributes и Span Attributes; см. Attributes для подробностей.
Sampler	Компонент стратегии, определяющий, следует ли сэмплировать и отправлять Trace. Можно настроить различные стратегии сэмплирования, такие как полное сэмплирование, пропорциональное и др.
ALB (Another Load Balancer)	Программное или аппаратное устройство, распределяющее сетевые запросы между доступными узлами в кластере; балансировщик нагрузки (ALB), используемый на платформе, является программным балансировщиком уровня 7, который можно настроить для мониторинга трафика с помощью OTel. ALB поддерживает отправку Trace на указанный Collector и позволяет использовать разные стратегии сэмплирования; также поддерживает настройку отправки Trace на уровне Ingress.
FT (Frontend)	Конфигурация порта для ALB, задающая настройки на уровне порта.

Термин	Объяснение
Rule	Правила маршрутизации на порту (FT), используемые для сопоставления конкретных маршрутов.
HotROD (Rides on Demand)	Пример приложения, предоставляемый Jaeger для демонстрации использования распределённого трассирования; подробности см. в Hot R.O.D. - Rides on Demand ↗.
hotrod-with-proxy	Указывает адреса внутренних микросервисов HotROD через переменные окружения; подробности см. в hotrod-with-proxy ↗.

Предварительные требования

- **Убедитесь, что существует работоспособный ALB**: Создайте или используйте существующий ALB, имя которого в данном документе заменено на `<otel-alb>`. Инструкции по созданию ALB см. в Deploy ALB.
- **Убедитесь, что имеется адрес сервера отчёта данных OTel**: Этот адрес далее будет называться `<jaeger-server>`.

Процедура

Обновление конфигурации ALB

1. На Master-узле кластера используйте CLI для выполнения команды редактирования конфигурации ALB:

```
kubectl edit alb2 -n cpaas-system <otel-alb> # Замените <otel-alb> на фактическое имя ALB
```

2. Добавьте следующие поля в раздел `spec.config`:

```
otel:
  enable: true
  exporter:
    collector:
      address: "<jaeger-server>" # Замените <jaeger-server> на фактический адрес сервера отчёта данных OTel
      request_timeout: 1000
```

Пример окончательной конфигурации:

```
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. Выполните команду для сохранения изменений:

```
:wq
```

После обновления ALB по умолчанию будет включён OpenTelemetry, и вся информация о Trace запросов будет отправляться на Jaeger Server.

Связанные операции

Настройка OTel в Ingress

- **Включение или отключение OTel на Ingress**

  Настройка включения OTel на Ingress позволяет лучше мониторить и отлаживать поток запросов приложений, выявляя узкие места производительности или ошибки путём трассировки запросов при их прохождении между сервисами.

  Процедура

  Добавьте следующую аннотацию в поле metadata.annotations Ingress:

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

  Объяснение параметра:

  - **nginx.ingress.kubernetes.io/enable-opentelemetry**: При значении `true` контроллер Ingress включает функциональность OpenTelemetry при обработке запросов через этот Ingress, что означает сбор и отправку информации о Trace запросов. При значении `false` или отсутствии этой аннотации сбор и отправка Trace не выполняются.

- **Включение или отключение доверия OTel на Ingress**

  OTel Trust определяет, доверяет ли Ingress и использует ли информацию Trace (например, trace ID) из входящих запросов.

  Процедура

  Добавьте следующую аннотацию в поле metadata.annotations Ingress:

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

  Объяснение параметра:

  - **nginx.ingress.kubernetes.io/opentelemetry-trust-incoming-span**: При значении `true` Ingress продолжает использовать уже существующую информацию Trace, что помогает поддерживать согласованность в межсервисном трассировании, позволяя полностью проследить и проанализировать всю цепочку запросов в системе распределённого трассирования. При значении `false` для запроса будет

сгенерирована новая информация трассировки, что может привести к тому, что запрос будет рассматриваться как часть новой цепочки трассировки после входа в Ingress, прерывая непрерывность межсервисного трассирования.

- **Добавление различных конфигураций OTel на Ingress**

  Эта настройка позволяет кастомизировать поведение OTel и методику экспорта данных для разных ресурсов Ingress, обеспечивая тонкий контроль над стратегией трассирования или целевым назначением для каждого сервиса.

  Процедура

  Добавьте следующую конфигурацию в поле metadata.annotations Ingress:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    alb.ingress.cpaas.io/otel: >
      {
         "enable": true,
         "exporter": {
             "collector": {
                 "address": "<jaeger-server>", # Замените <jaeger-server> на
фактический адрес сервера отчёта данных OTel, например "address":
"http://128.0.0.1:4318"
                 "request_timeout": 1000
             }
         }
      }
```

  Объяснение параметров:

  - **exporter**: Определяет способ отправки собранных данных Trace на OTel Collector (сервер отчёта данных OTel).
  - **address**: Адрес OTel Collector.
  - **request_timeout**: Таймаут запроса.

Использование OTel в приложениях

Ниже приведена полная структура конфигурации OTel, которая может использоваться для определения включения и использования возможностей OTel в приложениях.

На Master-узле кластера выполните команду CLI для получения полной структуры конфигурации OTel:

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

Результат:

```
{
    "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": ""
          },
      },
 }
```

Объяснение параметров:

Параметр	Описание
otel.enable	Включение функциональности OTel.
exporter.collector.address	Адрес сервера отчёта данных OTel, поддерживает протоколы http/https и доменные имена.
flags.hide_upstream_attrs	Отчёт информации о правилах upstream.
flag.notrust_incoming_span	Доверие и использование информации Trace OTel (например, trace ID) из входящих запросов.
flags.report_http_request_header	Отчёт заголовков запросов.
flags.report_http_response_header	Отчёт заголовков ответов.
sampler.name	Имя стратегии сэмплирования; подробности см. в Sampling Strategies.
sampler.options.fraction	Доля сэмплирования.
sampler.options.parent_name	Родительская стратегия для стратегий parent_base.

Наследование

По умолчанию, если в ALB настроены определённые параметры OTel, а в FT они не настроены, FT наследует параметры от ALB как свою конфигурацию; то есть FT наследует конфигурацию ALB, а Rule может наследовать конфигурации как от ALB, так и от FT.

- **ALB**: Конфигурация на ALB обычно глобальная и по умолчанию. Здесь можно настроить глобальные параметры, например адреса Collector, которые будут унаследованы нижестоящими FT и Rule.
- **FT**: FT может наследовать конфигурации от ALB, то есть параметры OTel, не настроенные в FT, будут использовать настройки ALB. При этом FT можно дополнительно уточнять; например, можно выборочно включать или отключать OTel на FT без влияния на другие FT или глобальные настройки ALB.

- **Rule**: Rule может наследовать конфигурации как от ALB, так и от FT. При этом Rule также можно уточнять; например, конкретное правило может отказаться от доверия входящей информации Trace OTel или изменить стратегии сэмплирования.

Процедура

Настройте поле `spec.config.otel` в YAML-файлах ALB, FT и Rule для добавления конфигурации, связанной с OTel.

Дополнительные сведения

Стратегии сэмплирования

Параметр	Объяснение
always on	Всегда отправлять все данные трассировки.
always off	Никогда не отправлять данные трассировки.
traceid-ratio	Решение об отправке принимается на основе `traceid`. Формат `traceparent` — `xx-traceid-xx-flag`, где первые 16 символов `traceid` представляют 32-битное шестнадцатеричное число. Если это число меньше `fraction`, умноженного на 4294967295 (то есть (2^32-1)), данные будут отправлены.
parent-base	Решение об отправке принимается на основе флага в traceparent запроса. При флаге 01 данные отправляются; например: `curl -v "http://$ALB_IP/" -H 'traceparent: 00-xx-xx-01'`. При флаге 02 данные не отправляются; например: `curl -v "http://$ALB_IP/" -H 'traceparent: 00-xx-xx-02'`.

Атрибуты

- **Resource Attributes**

  Эти атрибуты отправляются по умолчанию.

Параметр	Описание
hostname	Имя хоста Pod ALB
service.name	Имя ALB
service.namespace	Пространство имён, где расположен ALB
service.type	По умолчанию ALB
service.instance.id	Имя Pod ALB

- **Span Attributes**
  - Атрибуты, отправляемые по умолчанию:

Параметр	Описание
http.status_code	Код состояния
http.request.resend_count	Количество повторных попыток
alb.rule.rule_name	Имя правила, с которым совпал данный запрос
alb.rule.source_type	Тип правила, с которым совпал запрос, в настоящее время только Ingress
alb.rule.source_name	Имя Ingress
alb.rule.source_ns	Пространство имён, где расположен Ingress

  - Атрибуты, отправляемые по умолчанию, но могут быть исключены изменением поля flag.hide_upstream_attrs:

Параметр	Описание
alb.upstream.svc_name	Имя Service (внутреннего маршрута), на который перенаправляется трафик
alb.upstream.svc_ns	Пространство имён Service (внутреннего маршрута), на который перенаправляется трафик

Параметр	Описание
alb.upstream.peer	IP-адрес и порт Pod, на который перенаправляется трафик

- Атрибуты, не отправляемые по умолчанию, но могут быть отправлены изменением поля flag.report_http_request_header:

Параметр	Описание
`**http.request.header.<header>**`	Заголовок запроса

- Атрибуты, не отправляемые по умолчанию, но могут быть отправлены изменением поля flag.report_http_response_header:

Параметр	Описание
`**http.response.header.<header>**`	Заголовок ответа

Пример конфигурации

Ниже приведена YAML-конфигурация, которая разворачивает ALB и использует Jaeger в качестве сервера OTel, с Hotrod-proxy в качестве демонстрационного бэкенда. Настройка правил Ingress позволяет при запросах клиентов к ALB перенаправлять трафик на HotROD. Кроме того, взаимодействие между внутренними микросервисами HotROD также маршрутизируется через ALB.

1. Сохраните следующий YAML в файл с именем all.yaml.

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

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

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

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

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

2. В CLI выполните команду для развертывания Jaeger, ALB, HotROD и всех необходимых CR для тестирования.

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

3. Выполните команду для получения адреса доступа к Jaeger.

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

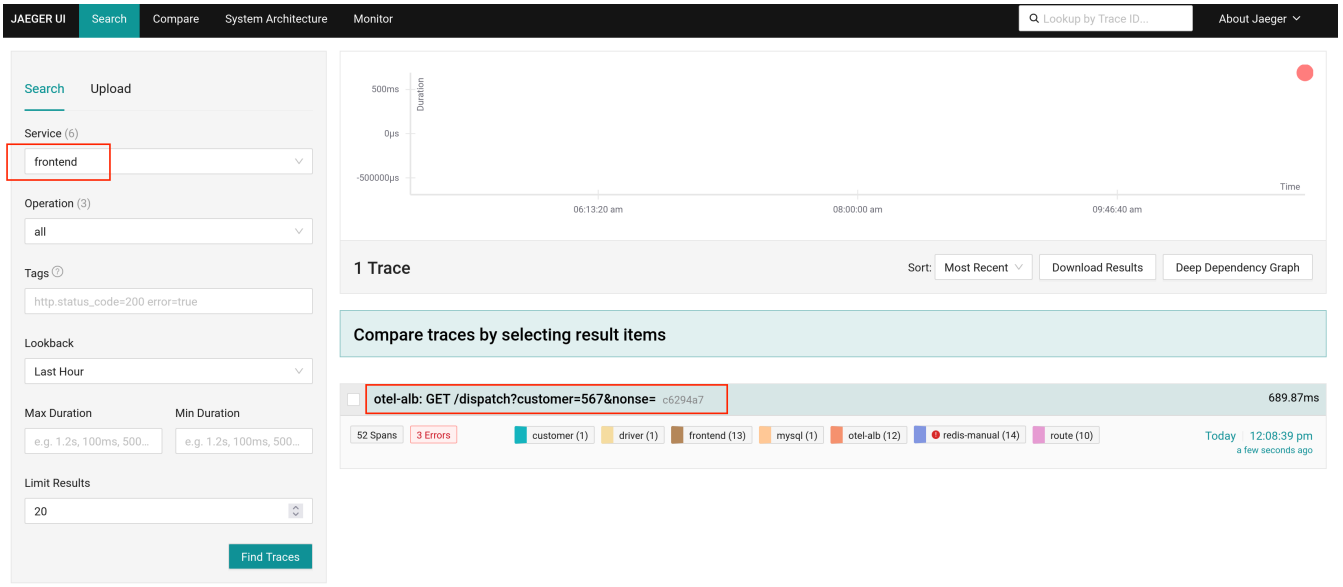
4. Выполните команду для получения адреса доступа к otel-alb.

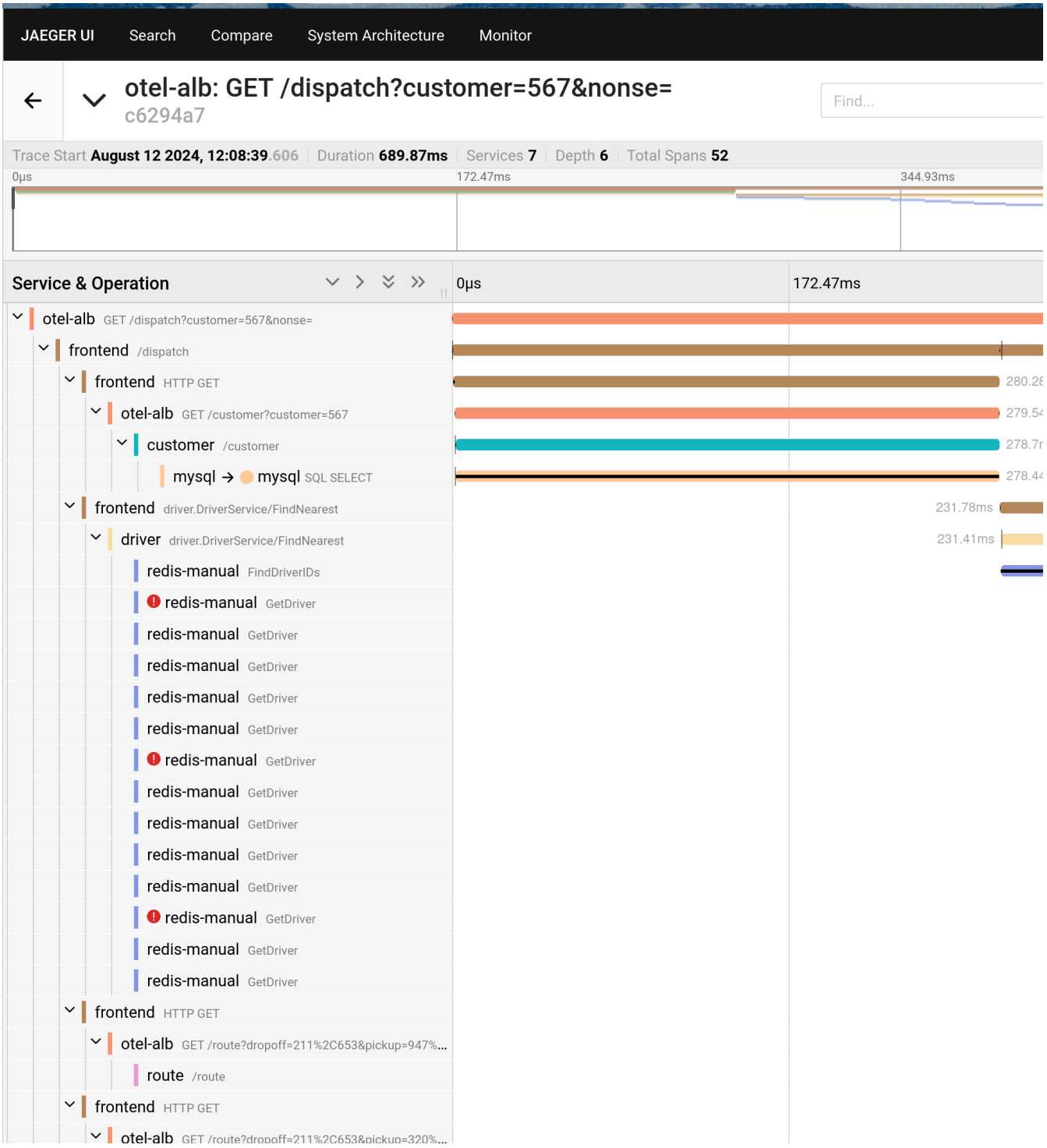
```
export ALB_IP=$(kubectl get po -A -o wide|grep otel-alb | awk '{print $7}');echo
$ALB_IP
```

5. Выполните команду для отправки запроса к HotROD через ALB. ALB при этом отправит Trace в Jaeger.

```
curl -v "http://<$ALB_IP>:80/dispatch?customer=567&nonse=" # Замените <$ALB_IP> в
команде на адрес доступа otel-alb, полученный на предыдущем шаге
```

6. Откройте адрес доступа Jaeger, полученный в Шаге 3, чтобы просмотреть результаты.




JAEGER UI
Search
Compare
System Architecture
Monitor
otel-alb: GET /dispatch?customer=567&nonse=
c6294a7
Find...
Trace Start August 12 2024, 12:08:39.606
Duration 689.87ms
Services 7
Depth 6
Total Spans 52
0µs
172.47ms
344.93ms
Service & Operation
0µs
172.47ms
otel-alb GET /dispatch?customer=567&nonse=
frontend /dispatch
frontend HTTP GET
otel-alb GET /customer?customer=567
customer /customer
mysql → mysql SQL SELECT
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

Содержание

Terminology

Term	Description
ALB	Самостоятельно разработанный платформой балансировщик нагрузки уровня 7.

Procedure

1. Перейдите в **Administrator**.
2. В левой навигационной панели нажмите **Operation Center** > **Monitoring** > **Monitoring Dashboard**.

3. В верхней части страницы нажмите **Cluster**, чтобы переключиться на нужный кластер для мониторинга.

4. Нажмите **Switch** в правом верхнем углу страницы.

5. Вы можете перейти на панель мониторинга **ALB Status** двумя способами:

   - Способ 1: Нажмите на карточку **container-platform**, чтобы раскрыть каталог мониторинга, затем нажмите на название **ALB Status**, чтобы войти в панель мониторинга. При необходимости вы можете установить эту панель мониторинга в качестве основной.

   - Способ 2: Введите ключевое слово (например, alb) в поле поиска и выполните поиск, затем нажмите на название **ALB Status**, чтобы войти в панель мониторинга. При необходимости вы можете установить эту панель мониторинга в качестве основной.

6. Просматривайте различные метрики мониторинга через панель.

   - **Выбор пространства имён для мониторинга**: Нажмите на **namespace** в верхней части страницы, чтобы выбрать пространство имён для мониторинга, по умолчанию — все, то есть мониторинг всех пространств имён.

   - **Выбор ALB для мониторинга**: Нажмите на **name** в верхней части страницы, чтобы выбрать ALB для мониторинга, по умолчанию — все, то есть мониторинг всех ALB.

Monitoring Metrics

Отображает метрики мониторинга общего трафика, использования ресурсов, Ingress (входящие правила), HTTPRoute (правила маршрутизации типа HTTPRoute) и Rule (правила, которые не являются ни Ingress, ни HTTPRoute) для выбранного ALB за **последние 5 минут**.

Примечание: Все данные — это данные мониторинга, собранные за **последние 5 минут**.

ALB Traffic Monitoring

Monitoring Metric	Description
Active Connections	Количество активных соединений на выбранном ALB.
Requests Per Second	Общее количество запросов, получаемых в секунду на выбранном ALB.
Error Rate	Доля запросов с ошибками 4XX (например, 404) и 5XX, возникающих в секунду на выбранном ALB.
Latency	Средняя задержка запросов на выбранном ALB.

ALB Resource Usage

Monitoring Metric	Description
CPU Usage	Использование CPU выбранного ALB.
Memory Usage	Использование памяти выбранного ALB.
Network Receive/Transmit	Пропускная способность сети выбранного ALB.
Disk Read/Write Rate	Пропускная способность диска выбранного ALB.

Ingress, HTTPRoute, Rule Traffic Monitoring

Monitoring Metric	Description
QPS (Queries Per Second)	Количество запросов в секунду, получаемых Ingress/HTTPRoute/Rule на выбранном ALB, единица измерения по умолчанию — req/s.
Request BPS (Bytes Per Second)	Общий размер запросов в байтах в секунду, получаемых Ingress/HTTPRoute/Rule на выбранном ALB.

Monitoring Metric	Description
Response BPS (Bytes Per Second)	Общий размер ответов в байтах, отправляемых Ingress/HTTPRoute/Rule на выбранном ALB.
Error Rate	Процент ошибок, возникших при обработке запросов Ingress/HTTPRoute/Rule на выбранном ALB.
P50, P90, P99	Время отклика запросов на выбранном ALB, а именно медианное время отклика. Это означает, что 50%, 90% и 99% запросов имеют время отклика меньше или равно этому значению. **Примечание**: Принцип P50, P90 и P99 заключается в сортировке собранных данных от меньшего к большему и взятии значений данных на позициях 50%, 90% и 99%; таким образом, 50%, 90% и 99% собранных данных находятся ниже этого значения. Процентили помогают анализировать распределение данных и выявлять различные крайние ситуации.
Upstream P50, Upstream P90, Upstream P99	Время отклика запросов к upstream-сервисам. Это означает, что 50%, 90% и 99% запросов, отправленных к upstream-сервисам, имеют время отклика меньше или равно этому значению.

CORS

Содержание

Основные понятия

CORS↗ (Cross-Origin Resource Sharing) — это механизм, который позволяет запрашивать ресурсы (например, шрифты, JavaScript и др.) на веб-странице с другого домена, отличного от домена, с которого изначально был получен ресурс.

CRD

```
enableCORS:
  description: enableCORS — переключатель, разрешающий кросс-доменные запросы,
    когда EnableCORS установлен в false, alb2 передаёт информацию на backend
    серверы, которые определяют, разрешать ли кросс-доменные запросы
  type: boolean
corsAllowHeaders:
  description: corsAllowHeaders определяет заголовки, разрешённые CORS,
    когда enableCORS установлен в true, несколько заголовков разделяются запятыми
  type: string
corsAllowOrigin:
  description: corsAllowOrigin определяет origin, разрешённые CORS,
    когда enableCORS установлен в true, несколько origin разделяются запятыми
  type: string
```

Это можно настроить в правиле `.spec`.

Политика сессионной аффинности балансировки нагрузки в ALB

В этом руководстве объясняются различные алгоритмы балансировки нагрузки, доступные в ALB, и как их настроить для оптимизации распределения трафика между подами вашего приложения.

Содержание

Обзор

ALB поддерживает несколько алгоритмов балансировки нагрузки для распределения входящего трафика между backend-подами. Выбор алгоритма зависит от требований

вашего приложения, таких как сохранение сессии, оптимизация производительности или равномерное распределение нагрузки.

Доступные алгоритмы

Round Robin (по умолчанию)

- **Политика**: `rr`
- **Описание**: Распределяет запросы последовательно по всем доступным подам по кругу.
- **Случай использования**: Лучший выбор для бессессионных приложений, где любой под может обработать любой запрос.

Source IP Hash

- **Политика**: `sip-hash`
- **Описание**: Последовательно направляет запросы с одного и того же IP-адреса источника к одному и тому же поду.
- **Поведение**: Использует первый IP из заголовка X-Forwarded-For, если он присутствует, иначе использует IP клиента.
- **Случай использования**: Полезно, когда требуется сохранение сессии на основе IP клиента.

Cookie-Based Affinity

- **Политика**: `cookie`
- **Атрибут**: `cookie-name` (по умолчанию `JSESSIONID`)
- **Описание**: Направляет запросы с одинаковым значением cookie к одному и тому же поду.
- **Поведение**:
  - Если указанный cookie отсутствует, ALB добавляет его в ответ

  - Формат cookie: `timestamp.worker_pid.random_number`
- **Случай использования**: Идеально для приложений, требующих привязки сессии на основе cookies.

Header-Based Affinity

- **Политика**: `header`
- **Атрибут**: `header-name`
- **Описание**: Направляет запросы с одинаковым значением заголовка к одному и тому же поду.
- **Случай использования**: Подходит для приложений, которым необходимо маршрутизировать на основе определённых HTTP-заголовков.

EWMA (Exponentially Weighted Moving Average)

- **Политика**: `ewma`
- **Описание**: Направляет трафик на основе времени отклика подов с использованием алгоритма Power of Two Choices (P2C) с EWMA.
- **Поведение**:
  - Поддерживает EWMA-оценку для каждого пода на основе времени отклика
  - Направляет трафик к подам с более низкими EWMA-оценками
  - Оценки экспоненциально затухают со временем
- **Случай использования**: Оптимально для приложений, чувствительных к задержкам
- **Ссылка**: Twitter Finagle EWMA Documentation ↗

Методы настройки

1. Использование аннотаций Ingress

```
annotations:
  alb.ingress.cpaas.io/session-affinity-policy: "<algorithm>"  # rr | sip-hash | cookie | header | ewma
  alb.ingress.cpaas.io/session-affinity-attribute: "<attribute>"  # Обязательно для политик cookie и header
```

2. Использование пользовательского ресурса ALB Frontend/Rule

```
spec:
  serviceGroup:
    session_affinity_policy: "<algorithm>"  # rr | sip-hash | cookie | header | ewma
    session_affinity_attribute: "<attribute>"  # Обязательно для политик cookie и header
```

Рекомендуемые практики

- Выбирайте Round Robin для бессессионных приложений с равномерным распределением запросов
- Используйте Source IP Hash, когда требуется сохранение сессии на основе IP клиента
- Реализуйте Cookie-based affinity для веб-приложений, требующих привязки сессии
- Рассматривайте EWMA для сервисов с переменным временем отклика для оптимизации задержек

Перезапись URL

Содержание

Основные понятия

ALB может переписывать URL запроса перед его передачей на backend. Вы можете использовать группы захвата в регулярных выражениях для переписывания URL.

Конфигурация

через аннотацию ingress

```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
```

через 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
```

Пример: клиент запрашивает `/prefix-x/abc`; backend получает `/abc`.

Calico Network поддерживает шифрование WireGuard

Calico поддерживает шифрование WireGuard как для IPv4, так и для IPv6 трафика, которое можно независимо включать с помощью параметров в ресурсе FelixConfiguration.

Содержание

Статус установки

Установка по умолчанию

Операционная система	Версия ядра
Linux	5.6 и выше установлены по умолчанию

Операционная система	Версия ядра
Ubuntu 20.04	5.4.0-135-generic
Kylin Linux Advanced Server V10 - SP3	4.19.90-52.22.v2207.ky10.x86_64

Не установлено по умолчанию

Операционная система	Версия ядра
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

Терминология

Термин	Объяснение
wireguardEnabled	Включить шифрование для IPv4 трафика поверх IPv4 Underlay сети.
wireguardEnabledV6	Включить шифрование для IPv6 трафика поверх IPv6 Underlay сети.

Примечания

1. При использовании сетевого плагина Calico убедитесь, что параметр `natOutgoing` установлен в `true` для поддержки шифрования WireGuard. По умолчанию этот параметр корректно настроен для подсети Calico при создании кластера и не требует дополнительной настройки.

2. WireGuard поддерживает шифрование как для IPv4, так и для IPv6 трафика; если необходимо шифровать оба типа трафика, настройка должна выполняться отдельно. Для подробной настройки параметров обратитесь к Felix Configuration Documentation ↗, настраивая параметры `wireguardEnabled` и `wireguardEnabledV6`.

3. Если WireGuard не установлен по умолчанию, обратитесь к WireGuard Installation Guide ↗ для ручной установки, хотя возможны случаи, когда ручная установка модуля WireGuard не удаётся.

4. Трафик между контейнерами на разных узлах будет зашифрован, включая сетевой трафик с одного хоста на другой; однако связь между Pod на одном узле и трафик между Pod и его хост-узлом не будет зашифрована.

Требования

- WireGuard должен быть установлен на всех узлах кластера заранее. Для подробностей обратитесь к WireGuard Installation Documentation ↗. Узлы без установленного WireGuard не поддерживают шифрование.

Процедура

1. Включите или отключите шифрование для IPv4 и IPv6.

   Примечание: Следующие команды необходимо выполнять в CLI-инструменте на Master-узле, где находится узел.

- Включить шифрование только для IPv4

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

- Включить шифрование только для IPv6

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

- Включить шифрование для IPv4 и IPv6

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

- Отключить шифрование для IPv4 и IPv6
  - Способ 1: Выполнить команду в CLI для отключения шифрования.

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

  - Способ 2: Изменить конфигурационный файл felixconfiguration для отключения шифрования.
    1. Выполните команду для открытия конфигурационного файла felixconfiguration.

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

    2. Установите параметры `wireguardEnabled` и `wireguardEnabledV6` в false для отключения шифрования WireGuard.

```
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 # Измените на true для включения шифрования
IPv4
  wireguardEnabledV6: false # Измените на true для включения шифрования
IPv6
```

2. После завершения настройки шифрования Calico WireGuard выполните следующую команду для подтверждения статуса шифрования WireGuard. Если включено шифрование IPv4 и IPv6, наличие `wireguardPublicKey` или `wireguardPublicKeyV6` в поле `Status` указывает на успешное включение; если шифрование IPv4 и IPv6 отключено, эти поля не будут содержать `wireguardPublicKey` или `wireguardPublicKeyV6`, что означает успешное отключение.

   ```
   calicoctl get node <NODE-NAME> -o yaml # Замените <NODE-NAME> на имя узла.
   ```

   Вывод:

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

Проверка результата

В этом документе приведён пример проверки трафика IPv4; проверка трафика IPv6 аналогична IPv4 и не повторяется.

Проверка трафика IPv4

1. После настройки шифрования WireGuard проверьте информацию о маршрутизации, где трафик между узлами преимущественно использует интерфейс wireguard.cali для пересылки сообщений.

```
root@test:~# ip rule   # Просмотр текущих правил маршрутизации
     0: from all lookup local
     99:  not from all fwmark 0x100000/0x100000 lookup 1    # Для всех пакетов без метки 0x100000 используется таблица маршрутизации 1
     32766:  from all lookup main
     32767 :  from all lookup default

root@test:~# ip route show table 1    # Отобразить записи маршрутизации для таблицы 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     # Трафик к IP 10.3.236.128 будет отправляться через интерфейс wireguard.cali
    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    # Маршрут от текущего узла до IP 10.10.10.202
    10.10.10.202 dev wireguard.cali table 1 src 10.10.10.127 uid 0  cache   # При доступе к IP 10.10.10.202 пакет будет отправлен через интерфейс wireguard.cali, используя таблицу маршрутизации 1, с исходным адресом 10.10.10.127

root@test:~# ip route    # Показать основную таблицу маршрутизации
    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 cali8dcd31cIdOO scope link
    dev cali3012b5b29b scope link
    dev calibeefea2ff87 scope link
```

```
    dev cali2b27d5e4053 scope link
    dev cali1a35dbdd639 scope link
    calico on link
```

2. Захват пакетов на узле для наблюдения трафика между узлами.

```
root@test:~# ip a s wireguard.cali    # Просмотр подробной информации об интерфейсе wireguard.cali
    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   # IP-адрес интерфейса wireguard.cali — 10.10.10.127
    valid_lft forever preferred_lft forever

root@test:~# tcpdump -i wireguard.cali -nnve icmp   # Захват и отображение ICMP пакетов через 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. Тестирование показывает, что трафик типа IPv4 пересылается через интерфейс wireguard.cali.

Kube-OVN Overlay Network поддерживает шифрование IPsec

В этом документе представлен подробный гид по включению и отключению шифрованного туннельного трафика IPsec в оверлейной сети Kube-OVN. Поскольку туннельный трафик OVN передаётся через физические маршрутизаторы и коммутаторы, которые могут находиться в ненадёжных публичных сетях или подвергаться атакам, включение шифрования IPsec эффективно предотвращает мониторинг и подмену данных трафика.

Содержание

Терминология

Термин	Объяснение
IPsec	Протокол и технология, используемые для защиты и проверки данных, передаваемых через интернет. Обеспечивает безопасную связь на уровне IP и в основном используется для создания виртуальных частных сетей (VPN) и защиты передачи IP-пакетов. IPsec

Термин	Объяснение
	обеспечивает безопасность данных преимущественно следующими способами: • **Шифрование данных**: с помощью технологий шифрования IPsec гарантирует, что данные не будут украдены или изменены во время передачи. Распространённые алгоритмы шифрования включают AES, 3DES и др. • **Проверка целостности данных**: IPsec использует хэш-функции (например, SHA-1, SHA-256) для проверки целостности данных, гарантируя, что данные не были изменены в процессе передачи. • **Аутентификация**: IPsec может проверять идентичность обеих сторон связи с помощью различных методов (например, предварительно разделённые ключи, цифровые сертификаты), чтобы предотвратить несанкционированный доступ. • **Управление ключами**: IPsec использует протокол Internet Key Exchange (IKE) для согласования и управления ключами шифрования.

Примечания

- Включение IPsec может вызвать кратковременное прерывание сети на несколько секунд.
- Если версия ядра — 3.10.0-1160.el7.x86_64, при включении функции IPsec в Kube-OVN могут возникнуть проблемы совместимости.

Предварительные требования

Выполните следующую команду, чтобы проверить, поддерживает ли текущее ядро операционной системы модули, связанные с IPsec. Если вывод показывает, что все

модули, связанные с XFRM, имеют значение `y` или `m`, это означает поддержку IPsec.

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

Вывод:

```
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
```

Процедура

Примечание: если не указано иное, все команды необходимо выполнять в CLI-инструменте на Master-узле кластера.

Включение IPsec

1. Измените конфигурационный файл kube-ovn-controller.
    1. Выполните следующую команду для редактирования YAML-конфигурации kube-ovn-controller.

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

    2. Измените указанные поля согласно следующим инструкциям.

```
spec:
  template:
    spec:
      containers:
        - args:
            - --enable-ovn-ipsec=true # Добавьте это поле
          securityContext:
            runAsUser: 0 # Измените значение на 0
```

Объяснение полей:

- **spec.template.spec.containers[0].args**: Добавьте `- --enable-ovn-ipsec=true` под этим полем.
- **spec.template.spec.containers[0].securityContext.runAsUser**: Измените значение этого поля на 0.

3. Сохраните изменения.

2. Измените конфигурационный файл kube-ovn-cni.

1. Выполните следующую команду для редактирования YAML-конфигурации kube-ovn-cni.

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

2. Измените указанные поля согласно следующим инструкциям.

```
spec:
  template:
    spec:
      containers:
        - args:
            - --enable-ovn-ipsec=true # Добавьте это поле
          volumeMounts: # Добавьте путь монтирования, смонтируйте том с
именем ovs-ipsec-keys в контейнер
            - mountPath: /etc/ovs_ipsec_keys
              name: ovs-ipsec-keys
      volumes: # Добавьте том с именем ovs-ipsec-keys типа hostPath
        - name: ovs-ipsec-keys
          hostPath:
            path: /etc/origin/ovs_ipsec_keys
```

Объяснение полей:

- **spec.template.spec.containers[0].args**: Добавьте `- --enable-ovn-ipsec=true` под этим полем.
- **spec.template.spec.containers[0].volumeMounts**: Добавьте путь монтирования и смонтируйте том с именем ovs-ipsec-keys в контейнер.
- **spec.template.spec.volumes**: Добавьте том с именем ovs-ipsec-keys типа hostPath под этим полем.

3. Сохраните изменения.

3. Проверьте, успешно ли функция была включена.

1. Выполните следующую команду для входа в Pod kube-ovn-cni.

```
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. Выполните следующую команду для проверки количества соединений Security Associations. Если их (число узлов - 1) в статусе up, это означает успешное включение.

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

Вывод:

```
Security Associations (2 up, 0 connecting):  # Поскольку в кластере 3 узла, видно, что количество соединений – 2 в статусе up
```

Отключение IPsec

1. Измените конфигурационный файл kube-ovn-controller.
    1. Выполните следующую команду для редактирования YAML-конфигурации kube-ovn-controller.

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

    2. Измените указанные поля согласно следующим инструкциям.

    ```
    spec:
      template:
        spec:
          containers:
            - args:
                - --enable-ovn-ipsec=false # Измените на false
              securityContext:
                runAsUser: 65534 # Измените значение на 65534
    ```

    Объяснение полей:

    - **spec.template.spec.containers[0].args**: Измените значение поля `enable-ovn-ipsec` на false.
    - **spec.template.spec.containers[0].securityContext.runAsUser**: Измените значение этого поля на 65534.

    3. Сохраните изменения.
2. Измените конфигурационный файл kube-ovn-cni.
    1. Выполните следующую команду для редактирования YAML-конфигурации kube-ovn-cni.

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

2. Измените указанные поля согласно следующим инструкциям.

- Конфигурация до изменения

```
spec:
  template:
    spec:
      containers:
        - args:
            - --enable-ovn-ipsec=true # Измените на false
          volumeMounts: # Удалите путь монтирования с именем ovs-ipsec-keys
            - mountPath: /etc/ovs_ipsec_keys
              name: ovs-ipsec-keys
      volumes: # Удалите том с именем ovs-ipsec-keys типа hostPath
        - name: ovs-ipsec-keys
          hostPath:
            path: /etc/origin/ovs_ipsec_keys
```

  Объяснение полей:

  - **spec.template.spec.containers[0].args**: Измените значение поля `enable-ovn-ipsec` на false.
  - **spec.template.spec.containers[0].volumeMounts**: Удалите путь монтирования с именем ovs-ipsec-keys.
  - **spec.template.spec.volumes**: Удалите том с именем ovs-ipsec-keys типа hostPath.

- Конфигурация после изменения

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

3. Сохраните изменения.

3. Проверьте, успешно ли функция была отключена.

   1. Выполните следующую команду для входа в Pod kube-ovn-cni.

      ```
      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. Выполните следующую команду для проверки статуса соединения. Если вывода нет, это означает успешное отключение.

      ```
      ipsec status
      ```

Menu

Устранение неполадок

Как решить проблемы межузловой коммуникации в ARM-средах?

Определение причины ошибки

Как решить проблемы межузловой коммуникации в ARM-средах?

При использовании более низких версий ядра и некоторых отечественных сетевых карт может возникать проблема некорректного вычисления контрольных сумм сетевой картой после включения Checksum Offload. Это может привести к сбоям в коммуникации между узлами в Kube-OVN Overlay сети. Конкретные решения следующие:

- **Решение 1: Обновление версии ядра**. Рекомендуется обновить версию ядра до 4.19.90-25.16.v2101 или выше.
- **Решение 2: Отключение Checksum Offload**. Если невозможно сразу обновить версию ядра и возникают проблемы с межузловой коммуникацией, можно отключить Checksum Offload для физической сетевой карты с помощью следующей команды.

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

Menu

Определение причины ошибки

Поле `X-ALB-ERR-REASON` в заголовке ответа на ошибочный запрос указывает причину ошибки.

Причина ошибки может быть следующей:

```
InvalidBalancer : no balancer found for xx # это означает, что для сервиса не найден конечный пункт
BackendError : read xxx byte data from backend # это означает, что backend действительно вернул ответ, ошибка не вызвана alb.
InvalidUpstream : no rule match # это означает, что запрос не соответствует ни одному правилу, поэтому alb возвращает 404.
```