Сетевые взаимодействия

Руководства

Настройка домена

Пример ресурса домена (CR)

Создание домена через веб-консоль

Создание домена с помощью CLI

Последующие действия

Дополнительные ресурсы

Создание сертификатов

Создание сертификата с помощью веб-

Настройка

Зачем нужен S

Пример Servic

Headless Servi

Настройка Ingress

Метод реализации

Пример Ingress:

Создание Ingress через веб-консоль

Настройка подсетей

Правила выделения IP

Сеть Calico

Сеть Kube-OVN

Управление подсетями

Configure MetalLB

Prerequisites

Configure an External IP Address Pool by using the web cons

ent

Настройка GatewayAPI Gateway

Предварительные требования

Настройка через веб-консоль

Настройка через YAML

Введение

Настройка

Предварительны

Настройка чер

Настройка чер

Следующий шаг

Введение

Следующий ша

зад

Настройка ALB

ALB

Frontend

Rule

Логи и мониторинг

Настройка NodeLocal DNSCache

Overview

Key Features

Important Notes

Installation

How It Works

Configuration

Configure C

Overview

Configuration

Как сделать

Задачи для Ingress-Nginx

Предварительные требования

Максимальное количество соединений

Таймаут запроса

Сессионная аффинность (Sticky Sessior

Модификация заголовков

Перезапись URL

HSTS (HTTP Strict Transport Security)

Ограничение скорости

WAF

Управление заголовком Forward

HTTPS

Сохранение исходного IP

Задачи для Envoy Gateway

Предварительные требования

Введение

Общие задачи для конфигурации Route

Расширенная конфигурация

Дополнительная конфигурация

alb

Soft Data C

Предварительны

Процедура

Проверка

Kube OVN

Overview

Key Features

Installation

How It Works

How To Activate

Uninstallation

Устранение неполадок

Как решить проблемы межуз*л* Определение причины ошибки

Руководства

Настройка домена

- Пример ресурса домена (CR)
- Создание домена через веб-консоль
- Создание домена с помощью CLI
- Последующие действия
- Дополнительные ресурсы

Создание сертификатов

- Создание сертификата с помощью веб-

Настройка

- Зачем нужен S
- Пример Servic
- Headless Servi

Настройка Ingress

- Метод реализации
- Пример Ingress:
- Создание Ingress через веб-консоль

Настройка подсетей

- Правила выделения IP
- Сеть Calico
- Сеть Kube-OVN
- Управление подсетями

Configure MetalLB

- Prerequisites
- Configure an External IP Address Pool by using the web cons

ent

Настройка GatewayAPI Gateway

- Предварительные требования
- Настройка через веб-консоль
- Настройка через YAML
- Введение
- Следующий шаг

Настройка

- Предварительны
- Настройка чер
- Настройка чер
- Введение
- Следующий ша

Настройка ALB

ALB

Frontend

Rule

Логи и мониторинг

Настройка NodeLocal DNSCache

Overview

Key Features

Important Notes

Installation

How It Works

Configuration

Configure C

Overview

Configuration

Настройка домена

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 разрешаться в 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 в 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 позволит использовать сертификат, связанный с доменным именем, для поддержки https.

Содержание

Пример ресурса домена (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 **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 открытый ключ (tls.crt) и закрытый 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 правильного 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 на другой сертификат 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 в проекте смогут использовать сертификат, импортированный и назначенный администратором платформы, при работе с входящими правилами и функционалом балансировки нагрузки. В дальнейшем, в таких ситуациях, как истечение срока действия сертификата, администратор платформы может централизованно обновить сертификат.

> **NOTE**
>
> Функционал 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 с помощью веб-консоли

1. Перейдите в раздел **Administrator**.
2. В левой навигационной панели выберите **Network Management** > **Certificates**.

3. Нажмите **Create Certificate**.

4. Ознакомьтесь с инструкциями 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, с помощью операции **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**.

Настройка сервисов

В Kubernetes Service — это способ открыть сетевое приложение, работающее в одном или нескольких Pod в вашем кластере.

Содержание

Зачем нужен Service

1. У Pod есть собственные IP, но:
   - IP Pod нестабильны (меняются при пересоздании Pod).
   - Прямой доступ к Pod становится ненадежным.
2. Service решает эту проблему, предоставляя:
   - Стабильный IP и DNS-имя.
   - Автоматическое балансирование нагрузки на соответствующие Pod.

Пример Service 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 Service, обычно определяется селектором, который вы задаёте.

3 Порт Service.

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

Headless Services

Иногда не требуется балансировка нагрузки и единый IP Service. В таком случае можно создать так называемые headless Services:

```
spec:
  clusterIP: None
```

Headless Services полезны, когда:

- Нужно обнаруживать IP отдельных Pod, а не только единый 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 навигационной панели выберите **Network** > **Services**.
3. Нажмите **Create Service**.
4. Следуйте инструкциям для настройки соответствующих параметров.

Параметр	Описание
Virtual IP Address	Если включено, для этого Service будет выделен ClusterIP, который можно использовать для обнаружения сервиса внутри кластера.

Параметр	Описание
	Если отключено, будет создан headless Service, обычно используемый для **StatefulSet**.
Type	• **ClusterIP**: Открывает Service на внутреннем IP кластера. При выборе этого значения Service доступен только внутри кластера. • **NodePort**: Открывает Service на IP каждого Node по статическому порту (NodePort). • **ExternalName**: Отображает Service на содержимое поля externalName (например, на hostname api.foo.bar.example). • **LoadBalancer**: Открывает Service снаружи с помощью внешнего балансировщика нагрузки. Kubernetes напрямую не предоставляет компонент балансировки нагрузки; его нужно предоставить самостоятельно или интегрировать кластер с облачным провайдером.
Target Component	• **Workload**: Service будет перенаправлять запросы на **конкретный** workload, который соответствует меткам, например, `project.cpaas.io/name: projectname` и `service.cpaas.io/name: deployment-name`. • **Virtualization**: Service будет перенаправлять запросы на **конкретную** виртуальную машину или группу виртуальных машин. • **Label Selector**: Service будет перенаправлять запросы на **определённый тип** workload с указанными метками, например, `environment: release`.
Port	Настройка сопоставления портов для этого Service. В приведённом примере другие Pod внутри кластера могут обращаться к этому Service по виртуальному IP (если включён) и TCP-порту *80*; запросы будут

Параметр	Описание
	перенаправлены на внешний TCP-порт *6379* или *redis* Pod целевого компонента. • **Protocol**: Протокол, используемый Service, поддерживаются: `TCP`, `UDP`, `HTTP`, `HTTP2`, `HTTPS`, `gRPC`. • **Service Port**: Номер порта, который Service открывает внутри кластера, то есть Port, например, *80*. • **Container Port**: Целевой порт (или имя), на который маппится service port, 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 привязывается к исходному IP-адресу (ClientIP). Если включено, все запросы с одного 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
```

Создание сервиса 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 Kubernetes автоматически: <servic
e-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!".

Конечно, можно получить доступ к приложению снаружи кластера, создав Service типа LoadBalancer.

Примечание: Пожалуйста, заранее настройте сервис LoadBalancer.

```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   80:3000
5/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: Service 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 имя разрешается в `example.com`.

Аннотации для Service типа LoadBalancer

AWS EKS Cluster

Подробное описание аннотаций для LoadBalancer Service в EKS смотрите в 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 отправляться на Pod через NodePort. • ip: Трафик направляется напрямую на Pod (кластер должен использовать Amazon VPC CNI).	Определяет, как трафик достигает Pod.

Ключ	Значение	Описание
service.beta.kubernetes.io/aws-load-balancer-scheme	• internal: Частная сеть. • internet-facing: Публичная сеть.	Определяет использование частной или публичной сети.
service.beta.kubernetes.io/aws-load-balancer-ip-address-type	• IPv4 • dualstack	Определяет поддерживаемый стек IP-адресов.

Huawei Cloud CCE Cluster

Подробное описание аннотаций для LoadBalancer Service в CCE смотрите в Annotation Usage Documentation ↗ .

Ключ	Значение
kubernetes.io/elb.id	
kubernetes.io/elb.autocreate	Пример: `{"type":"public","bandwidth_name":"cce-bandwidth-1551163379627","bandwidth_chargemode":"bandwidth","bandwi["cn-north-4b"],"l4_flavor_name":"L4_flavor.elb.s1.small"` **Примечание**: Пожалуйста, ознакомьтесь с Инструкцией по запо
kubernetes.io/elb.subnet-id	

Ключ	Значение
kubernetes.io/elb.class	• union: Общая балансировка нагрузки. • performance: Эксклюзивная балансировка нагрузки, поддерж
kubernetes.io/elb.enterpriseID	

Azure AKS Cluster

Подробное описание аннотаций для LoadBalancer Service в AKS смотрите в Annotation Usage Documentation ↗ .

Ключ	Значение	Описание
service.beta.kubernetes.io/azure-load-balancer-internal	• true: Частная сеть. • false: Публичная	Определяет использование частной или публичной сети.

Ключ	Значение	Описание
		сеть.

Google GKE Cluster

Подробное описание аннотаций для LoadBalancer Service в GKE смотрите в Annotation Usage Documentation ↗ .

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

Пример: LoadBalancer с MetalLB BGP и Local Traffic Policy

В этом примере показано, как настроить Service типа LoadBalancer с использованием MetalLB в режиме BGP и параметром `externalTrafficPolicy: Local` для реализации активного активного балансирования нагрузки без дополнительных сетевых переходов.

Преимущества

- **Активное активное балансирование нагрузки**: Трафик распределяется одновременно по нескольким узлам
- **Отсутствие дополнительных сетевых переходов**: Прямой маршрут к Pod без промежуточной пересылки через узлы
- **Лучшее быстродействие**: `externalTrafficPolicy: Local` сохраняет исходный IP и снижает задержки

- **Высокая доступность**: Объявления маршрутов BGP обеспечивают доставку трафика до здоровых узлов

Предварительные требования

Перед настройкой LoadBalancer Service убедитесь, что у вас:

1. **Развернут MetalLB**: См. Создание пула внешних IP-адресов
2. **Настроен BGP Peer**: См. Создание BGP Peer
3. **Пул внешних IP-адресов**: Настроен IPAddressPool с BGPAdvertisement

Шаги

Разверните приложение с Service типа LoadBalancer и `externalTrafficPolicy: Local`:

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

Ключевые моменты настройки

externalTrafficPolicy: Local

Параметр `externalTrafficPolicy: Local` обеспечивает:

- **Сохранение исходного IP**: Исходный IP клиента сохраняется, что важно для логирования и политики безопасности
- **Прямой маршрут к Pod**: Трафик направляется напрямую к Pod без пересылки на уровне узла

LoadBalancer с BGP

При использовании MetalLB в режиме BGP:

- Маршруты объявляются с узлов, указанных в nodeSelectors BGPAdvertisement
- BGP peer получает эти объявления и маршрутизирует трафик соответствующим образом
- Совпадение селекторов узлов между BGPPeer и BGPAdvertisement обеспечивает согласованность маршрутизации

Шаги развертывания

1. **Разверните приложение**:

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

2. **Проверьте Service LoadBalancer**:

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

Ожидаемый вывод:

```
NAME                        TYPE           CLUSTER-IP      EXTERNAL-IP   P
ORT(S)        AGE
nginx-loadbalancer-local LoadBalancer   10.0.2.57       4.4.4.3       8
0:30005/TCP   30s
```

3. **Проверьте работу сервиса**:

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

Проверка

- **Мониторинг endpoints сервиса**: `kubectl get endpoints nginx-loadbalancer-local`
- **Просмотр статуса сервиса**: `kubectl describe svc nginx-loadbalancer-local`

Настройка 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

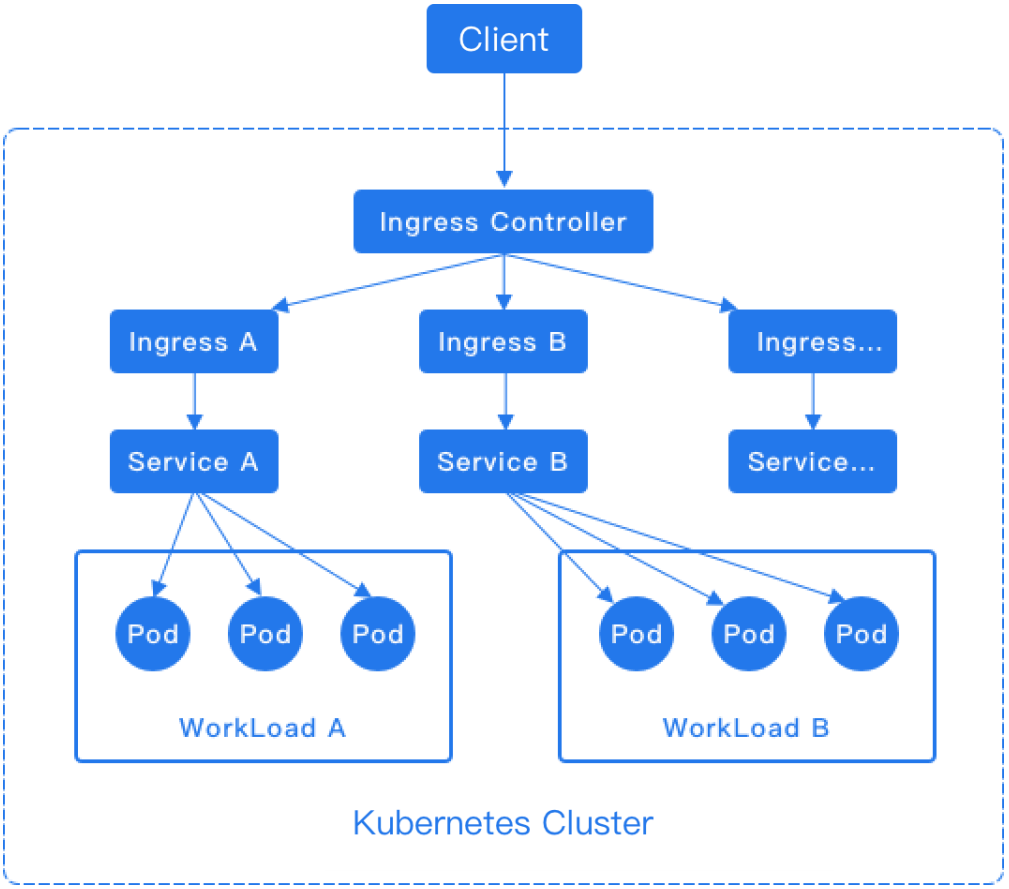
При создании нескольких ingress в одном namespace разные ingress **НЕ ДОЛЖНЫ** иметь одинаковые **Domain**, **Protocol** и **Path** (то есть дублирование точек доступа не допускается).

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 в качестве ingress-controller через ingress-nginx-operator
- Установка alb в качестве ingress-controller через alb-operator

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 дополнительных настроек обратитесь к nginx-configuration ↗.

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

3 Если вы хотите запускать ingress только 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 этого параметра.
Domain Name	Хосты могут быть точным совпадением (например, `foo.bar.com`) или шаблоном с подстановочным знаком (например, `*.foo.com`). Доступные доменные имена выделяются администратором платформы.
Certificates	TLS секрет 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
```

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 этой конкретной 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 этой конкретной 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, создаваемые в сети Calico, по умолчанию используют инкапсуляцию VXLAN. Порты, необходимые для инкапсуляции VXLAN, отличаются от портов для инкапсуляции IPIP. Необходимо убедиться, что открыт UDP порт 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. При `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 выберите **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 данного CIDR для связи. **Примечание**: Соответствие между CIDR и BlockSize смотрите 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**: всегда включать IPIP / VXLAN туннели. • **Cross Subnet**: включать IPIP / VXLAN туннели 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 включён. Используется для задания адресов доступа, которые будут видны внешней сети при выходе контейнерных групп подсети во внешний мир. Если NAT включён, в качестве адреса доступа используется IP хоста; если выключен — 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 приведено в таблице ниже.

CIDR	Размер blockSize	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	Размер blockSize	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

Подсети с префиксом больше 31 не поддерживаются.

Сеть Kube-OVN

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

NOTE

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

Пример custom resource (CR) подсети с Overlay-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 Overlay-сети Kube-OVN через веб-консоль

1. Перейдите в **Administrator**.
2. В левой навигационной панели выберите **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 подсети для управления 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 облегчает 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 обрабатывает трафик. При включении ECMP (Equal-Cost Multipath Routing) исходящий трафик маршрутизируется по нескольким равнозначным путям ко всем доступным узлам шлюза, увеличивая общую пропускную способность. **Примечание**: необходимо заранее 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 включён. Используется для задания адреса доступа, видимого во внешней сети при выходе Pods подсети в интернет. Если NAT включён, в качестве адреса доступа используется IP хоста;

Параметр	Описание
	если выключен — 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 Overlay-сети Kube-OVN через CLI

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

Underlay-сеть

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 со стороны физической сети. Рекомендуется ознакомиться с лучшими практиками в разделе Preparing the Kube-OVN Underlay Physical Network для обеспечения сетевой связности.

Инструкция по использованию

Общий процесс создания подсетей в Underlay-сети Kube-OVN: 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 передавать трафик контейнерной сети, обеспечивая связь с физической сетью.

Процедура:

1. Перейдите в **Administrator**.
2. В левой навигационной панели выберите **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 конкретным подсетям, запланированные на текущем узле. Это зависит от области действия подсети под мостовой сетью.
   - Узлы в Underlay-подсети должны иметь несколько сетевых карт, и сетевая карта, используемая мостовой сетью, должна быть выделена исключительно для Underlay и не должна использоваться для другого трафика, например SSH.

Например, если в мостовой сети три узла, планирующие eth0, eth0, eth1 для эксклюзивного 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 вместо сетевой карты 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 других узлов, 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 bridge network.

Платформа имеет преднастроенную виртуальную LAN **ovn-vlan**, которая подключается к bridge network **provider**. Вы также можете создать новую VLAN, подключенную к другим bridge network, обеспечивая изоляцию между VLAN.

Процедура:

1. Перейдите в **Administrator**.
2. В левой навигационной панели выберите **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 подключена к этой bridge network для связи с физической сетью.

5. Нажмите **Add**.

Добавление VLAN через CLI

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

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

```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 Underlay-сети Kube-OVN через веб-консоль

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

Процедура:

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

4. Настройте параметры согласно следующим инструкциям.

Параметр	Описание
VLAN	VLAN, к которой 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 выделяться. Например, могут использоваться как фиксированные IP 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 Underlay-сети Kube-OVN через CLI

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

Связанные операции

Если в кластере существуют подсети как Underlay, так и Overlay, можно при необходимости настроить Автоматическую взаимосвязь между Underlay и Overlay подсетями.

Управление подсетями

Обновление шлюза через веб-консоль

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

1. Перейдите в **Administrator**.
2. В левой боковой панели выберите **Network Management** > **Subnets**.
3. Нажмите на имя подсети.
4. Выберите **Action** > **Update Gateway**.
5. Обновите параметры; подробности смотрите 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": "192.168.66.21
0"},
  {"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 боковой панели выберите **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 боковой панели выберите **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 боковой панели выберите **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 использования 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 боковой панели выберите **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 смотрите 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 смогут продолжать работу 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 выберите **Network Management** > **Subnets**.
3. Нажмите ⋮ > **Delete** и подтвердите удаление.

Удаление подсети через CLI

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

Configure MetalLB

Содержание

Prerequisites

Пожалуйста, убедитесь, что вы прочитали документацию по Installation перед продолжением.

Configure an External IP Address Pool by using the web console

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 приведённым ниже инструкциям для настройки параметров.

Parameter	Description
Type	• L2: Связь и пересылка на основе MAC-адресов, подходит для небольших или локальных сетей, требующих простого и быстрого 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-адресов. Нажмите **Add** для добавления нескольких записей, примеры: **CIDR**: `192.168.1.1/24`. **IP Range**: `192.168.2.1` ~ `192.168.2.255`.
Available Nodes	В режиме L2 доступные узлы — это узлы, через которые проходит весь трафик VIP; в режиме BGP доступные узлы — это узлы, которые несут VIP, устанавливают BGP-соединения с пирингами и объявляют маршруты внешне. • **Node Name**: Выбор доступных узлов по имени узла. • **Label Selector**: Выбор доступных узлов по меткам. • **Show Node Details**: Просмотр итогового списка доступных узлов. **Примечание**: • При использовании типа BGP доступные узлы являются следующими хопами; убедитесь, что выбранные доступные узлы являются подмножеством узлов BGP Connection Nodes. • Можно настроить либо селектор меток, либо имя узла для выбора доступных узлов; если настроены оба параметра одновременно, итоговые доступные узлы — пересечение обоих.

Parameter	Description
BGP Peers	Выберите BGP-пиров; подробности настройки смотрите в разделе BGP Peers.

5. Нажмите **Create**.

Configure BGP Peers by using the web console

1. Перейдите в **Administrator**.
2. В левой навигационной панели выберите **Network Management** > **BGP Peers**.
3. Нажмите **Create BGP Peer**.
4. Следуйте инструкциям ниже для настройки параметров.

Parameter	Description
Local AS Number	Номер AS автономной системы, в которой находится узел с BGP-соединением. **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 автономной системы, в которой находится BGP-пир.
Peer IP	IP-адрес BGP-пира, должен быть действительным IP-адресом, способным установить BGP-соединение.
Local IP	IP-адрес узла с BGP-соединением. Если у узла несколько IP, выберите конкретный локальный IP для установления BGP-соединения с пиром.
Peer Port	Номер порта BGP-пира.

Parameter	Description
BGP-Connected Node	Узел, который устанавливает BGP-соединение. Если параметр не задан, BGP-соединения будут установлены со всеми узлами.
eBGP Multi-Hop	Позволяет устанавливать BGP-сессии между BGP-маршрутизаторами, которые не связаны напрямую. При включении этой функции значение TTL BGP-пакетов по умолчанию равно 5, что позволяет устанавливать пиринговые отношения BGP через несколько промежуточных сетевых устройств, делая дизайн сети более гибким.
RouterID	32-битное числовое значение (обычно в формате с точками, похожем на IPv4-адрес), используемое для уникальной идентификации BGP-маршрутизатора в сети BGP, обычно применяется для установления соседских отношений BGP, обнаружения петель маршрутизации, выбора оптимальных путей и устранения неполадок сети.

5. Нажмите **Create**.

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

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

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
```

```
kubectl apply -f ippool-with-L2advertisement.yaml -f ippool-with-bgpadver
tisement.yaml
```

Troubleshooting MetalLB

Symptom	Possible Cause	Resolution
No external IP assigned	Нет действительного IPAddressPool или неправильная конфигурация пула	Проверьте диапазон IP и namespace

Symptom	Possible Cause	Resolution
Pods CrashLoop	Отсутствуют RBAC для Speaker или Controller	Проверьте разрешения оператора
BGP not established	Несовпадение ASN или пир недоступен	Проверьте спецификацию `BGPPeer` и сетевые маршруты
L2 not working	Неправильный VLAN или фильтрация ARP	Используйте `arping` для проверки доступности широковещательной рассылки

Для получения дополнительной информации смотрите Troubleshooting MetalLB ↗

Настройка GatewayAPI Gateway

Содержание

Предварительные требования

Пожалуйста, убедитесь, что вы прочитали документацию по установке перед продолжением.

Настройка через веб-консоль

1. Перейдите в `Alauda Container Platform -> Networking -> Gateways`
2. Нажмите кнопку `Create Gateway`
3. На странице `Create Gateway` выберите `envoy-gateway-operator-cpaas-default` в поле GatewayClass, после чего отобразятся следующие параметры конфигурации:

Поле	Описание	Путь в YAML
Name	имя	gateway: `.metadata.name` envoygateway: `.metadata.name`
GatewayClass	какой GatewayClass используется	gateway: `.spec.gatewayClassName`
Service Type	тип сервиса	envoygateway: `.spec.provider.kubernetes.envoyServi`
Service Annotation	аннотация сервиса	envoygateway: `.spec.provider.kubernetes.envoyService.annotation`
Resource Limits	лимиты ресурсов деплоймента	envoygateway: `.spec.provider.kubernetes.envoyDeployment.contain`
Replicas	количество реплик деплоймента	envoygateway: `.spec.provider.kubernetes.envoyServi`
Node Labels	метки узлов деплоймента	envoygateway: `.spec.provider.kubernetes.envoyService.nodeSelect`

Поле	Описание	Путь в YAML
Listener	listener	gateway: `.spec.listeners`

WARNING

Форма веб-консоли поддерживает только GatewayClasses, созданные с помощью `EnvoyGatewayCtl`. Для других GatewayClasses используйте редактор YAML.

NOTE

При использовании `GatewayClass`, созданного `EnvoyGatewayCtl`, веб-консоль автоматически создаёт сопутствующий ресурс envoyproxy с именем и пространством имён, совпадающими с Gateway.

Настройка через YAML

```yaml
apiVersion: gateway.networking.k8s.io/v1
kind: Gateway
metadata:
  name: demo
  namespace: demo
spec:
  infrastructure: (1)
    parametersRef:
      group: gateway.envoyproxy.io
      kind: EnvoyProxy
      name: demo
  gatewayClassName: envoy-gateway-operator-cpaas-default (2)
  listeners: (3)
    - name: http
      port: 80
      hostname: a.com (4)
      protocol: HTTP (5)
      allowedRoutes:
        namespaces:
          from: Same (6)
    - name: https
      port: 443
      hostname: a.com
      protocol: HTTPS
      allowedRoutes:
        namespaces:
          from: Same
      tls: (7)
        mode: Terminate
        certificateRefs:
          - name: demo-tls
    - name: tcp
      port: 8080
      protocol: TCP
      allowedRoutes:
        namespaces:
          from: Same
    - name: udp
      port: 8080
      protocol: UDP
      allowedRoutes:
        namespaces:
          from: Same
```

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

1. Сопутствующий envoyproxy
2. Указание, к какому `GatewayClass` относится
3. Listener
4. Hostname слушателя
5. Поддерживаемый тип слушателя
6. Разрешённые namespace для маршрутов
7. Конфигурация TLS
8. Имя сопутствующего envoyproxy
9. Тип сервиса
10. Пожалуйста, сохраните и не изменяйте значение по умолчанию репозитория
11. Ресурсы для экземпляра `envoy-proxy`
12. Пожалуйста, сохраните и не изменяйте

Введение

Тип сервиса

Тип сервиса определяет, как будет открыт доступ к шлюзу. Существует три режима: LoadBalancer, NodePort и ClusterIP.

LoadBalancer (Рекомендуется)

Преимущество — простота использования и возможности высокодоступного балансировщика нагрузки. Для использования LoadBalancer в кластере должна быть поддержка LoadBalancer, которую можно включить через MetalLB.

NodePort

Преимущество — отсутствие внешних зависимостей.

Однако у NodePort есть следующие недостатки:

- Можно использовать только в кластерах с количеством узлов менее 16, иначе статус шлюза может стать некорректным.
- При использовании NodePort Kubernetes назначает номера портов NodePort, отличающиеся от портов сервиса. Для доступа необходимо использовать порт NodePort, а не порт сервиса.
- Сервис доступен по IP-адресу любого узла кластера, что может представлять потенциальные риски безопасности.

Как получить правильный порт при использовании NodePort

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

Выводом будет порт NodePort.

ClusterIP

Очень удобно, если внешний доступ не требуется.

Listener

Listener определяет порт и протокол, на которых слушает шлюз. Для протоколов HTTP или HTTPS разные hostnames могут рассматриваться как разные слушатели.

Нельзя создавать слушателя с конфликтующим портом, протоколом или hostname.

В `Gateway` должен быть создан хотя бы один listener.

Поддерживаемый тип слушателя

Каждый listener поддерживает разные типы маршрутов в зависимости от протокола:

Протокол слушателя	Поддерживаемый тип маршрута
HTTP	HTTPRoute
HTTPS	HTTPRoute, GRPCRoute
TLS	TLSRoute
TCP	TCPRoute
UDP	UDPRoute

При настройке маршрутов убедитесь, что они соответствуют протоколу слушателя, к которому будут прикреплены. Например, нельзя прикрепить HTTPRoute к TCP listener.

Разрешённые namespace для маршрутов

По умолчанию маршруты могут прикрепляться только к Gateway в том же namespace. Чтобы разрешить маршрутизацию между namespace, необходимо указать, какие namespace разрешены для прикрепления маршрутов к listener этого Gateway с помощью поля `allowedRoutes`.

Для подробностей смотрите attach to gateway create on other ns.

TLS

По умолчанию можно использовать только `secret`, созданный в том же namespace. В противном случае смотрите use secret create on other ns.

По умолчанию используется режим `Terminate`. Для других режимов смотрите ssl passthrough.

Hostname

Hostname в listener является уникальным идентификатором для слушателей с одинаковым протоколом. Нельзя добавить или обновить слушателя с конфликтующим hostname в одном Gateway.

Правила пересечения hostname слушателя и hostname маршрутов

При поступлении запроса он сопоставляется с **пересечением** hostname слушателя и hostname маршрутов. Для маршрутизации используются только hostname из пересечения.

Hostname слушателя	Hostname маршрутов	Результат пересечения	Пример
Нет hostname	Нет hostname	Совпадает со всеми хостами	Любой входящий заголовок host принимается
Нет hostname	Есть hostname (например, `api.example.com`)	Все hostname маршрутов	Совпадают только запросы с `api.example.com`
Есть hostname (например, `api.example.com`)	Нет hostname	Все hostname слушателя	Совпадают только запросы с `api.example.com`
Есть hostname (например, `api.example.com`)	Есть точное совпадение hostname	Точное совпадение hostname	Совпадают только запросы с `api.example.com`

Hostname слушателя	Hostname маршрутов	Результат пересечения	Пример
	(например, `api.example.com`)		
Есть wildcard (например, `*.example.com`)	Есть совпадающие hostname (например, `api.example.com`, `web.example.com`)	Совпадающие конкретные hostname	Совпадают запросы с `api.example.com` или `web.example.com`
Есть hostname (например, `api.example.com`)	Есть несовпадающие hostname (например, `web.example.com`)	**Нет пересечения** — статус маршрута некорректен	Маршрут не может обрабатывать трафик

NOTE

Подстановочные знаки (`*`) выполняют сопоставление по суффиксу. Например, `*.example.com` соответствует `foo.example.com` и `bar.example.com`, но не `example.com`.

WARNING

Отсутствие пересечения — статус маршрута некорректен, маршрут не может обрабатывать трафик.

Сопутствующий EnvoyProxy

`Envoy Gateway` предоставляет различные уровни контроля над развертыванием шлюзов. Рекомендуется создавать отдельный ресурс `EnvoyProxy` для каждого Gateway и ссылаться на него через поле `.spec.infrastructure.parametersRef` Gateway.

Такой подход один к одному обеспечивает лучшую изоляцию и более тонкий контроль над конфигурациями развертывания, такими как количество реплик, ресурсы и

ограничения планирования.

Для других способов конфигурации развертывания смотрите deployment-mode↗.

Репозиторий образов

Это значение по умолчанию. Оно будет заменено на фактический репозиторий образов текущего кластера, пожалуйста, не изменяйте его.

Следующий шаг

Настройка GatewayAPI Route

Настройка маршрута GatewayAPI

Содержание

Предварительные требования

Пожалуйста, убедитесь, что вы ознакомились с документацией по установке перед продолжением.

Настройка через веб-консоль

1. Перейдите в `Alauda Container Platform -> Networking -> Routes`
2. Нажмите кнопку `Create Route`

Создание HTTPRoute

Поле	Описание	Путь в YAML
Publish to Listener	публикация на слушатель	`.spec.parentRefs`
Hostnames	имена хостов	`.spec.hostnames`
Matches	совпадения	`.spec.rules[].matches`
Filters	фильтры	`.spec.rules[].filters`
Backend Instance	бэкенд	`.spec.rules[].backendRefs`
Session Affinity	сессионная аффинность	`.spec.rules[].sessionPersistence`

Создание TCP/UDP маршрута

Поле	Описание	Путь в YAML
Publish to Listener	публикация на слушатель	`.spec.parentRefs`
Backend Instance	бэкенд	`.spec.rules[].backendRefs`

Настройка через YAML

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

```
        - group: ''
          kind: Service
          name: echo-resty
          port: 80
          weight: 100
---
apiVersion: gateway.networking.k8s.io/v1alpha2
kind: UDPRoute
metadata:
  name: udp
  namespace: demo
spec:
  parentRefs:
    - group: gateway.networking.k8s.io
      kind: Gateway
      name: demo
      namespace: demo
      sectionName: udp
  rules:
    - backendRefs:
        - group: ''
          kind: Service
          name: echo-resty
          namespace: demo
          port: 53
          weight: 100
```

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

Введение

Каждый маршрут — это CR, определённый спецификацией `GatewayAPI`. Для подробной информации о полях и параметрах настройки каждого типа маршрута, пожалуйста, обратитесь к официальной документации: - HTTPRoute specification ↗ - TCPRoute specification ↗ - UDPRoute specification ↗ - GRPCRoute specification ↗ - TLSRoute specification ↗

Hostnames

Поле `hostnames` в маршруте представляет собой массив строк.
Оно следует правилам пересечения имён хостов.

Publish to Listener

- `sectionName` — это имя слушателя.
- Маршруты могут быть прикреплены только к слушателям, поддерживающим их конкретный тип.
- По умолчанию маршруты могут быть прикреплены только к слушателям, где `Gateway` находится в том же namespace. Для прикрепления между namespace смотрите раздел прикрепление к gateway, созданному в другом namespace.

Rules

Каждый маршрут может содержать несколько правил. Каждое правило состоит из следующих компонентов:

Matches

Определяет условия, которые должны быть выполнены, чтобы запрос был маршрутизирован этим правилом.

Правило может содержать несколько совпадений:

- Каждое совпадение состоит из нескольких условий (например, путь, заголовки, параметры запроса, метод)
- Условия **внутри** одного совпадения используют логику **AND** (все должны быть выполнены)

- Совпадения **между собой** используют логику **OR** (достаточно выполнить любое совпадение)

Пример: Если Match-1 требует `path=/api` И `header=v1`, а Match-2 требует `query=test`, то запрос будет маршрутизирован, если он соответствует либо `(path=/api И header=v1)`, либо `(query=test)`.

Типы условий совпадения

Объект	Метод	Типы значений	Описание	Требования к значению
Path				
	`Exact`	путь (строка)	Совпадение с точным URL-путём с учётом регистра. Это означает, что точное совпадение пути /abc сработает только для запросов к /abc, **НЕ** для /abc/, /Abc или /abcd.	Должен начинаться с без последовательн `//`.
	`PathPrefix`	путь (строка)	Совпадение по префиксу URL-пути, разделённому по /. Совпадение чувствительно к регистру и происходит **поэлементно**. Например, пути /abc,	Должен начинаться с без последовательн `//`.

Объект	Метод	Типы значений	Описание	Требования к значению
			/abc/ и /abc/def совпадут с префиксом /abc, а путь /abcd — нет.	
	`RegularExpression`	путь (строка)	Регулярное выражение: движок RE2.	например, `/api/v1/.*`.
Header				
	`Exact`	имя (ключ заголовка) + значение	Точное совпадение значения заголовка.	
	`RegularExpression`	имя (ключ заголовка) + значение	Регулярное выражение: движок RE2.	
QueryParam				
	`Exact`	имя (ключ параметра) + значение	Точное совпадение значения параметра запроса.	Значение параметра: 1-1 символа
	`RegularExpression`	имя (ключ параметра) + значение	Регулярное выражение: движок RE2.	

Объект	Метод	Типы значений	Описание	Требования к значению
Method	-	имя метода	Совпадение HTTP-метода.	`GET`, `HEAD`, `POST`, `PUT`, `DELETE`, `CONNECT`, `OPTIONS`, `TRACE`, `PATC`

Ссылки на условия совпадения

Тип условия	Официальная документация
Path	HTTPPathMatch ↗
Headers	HTTPHeaderMatch ↗
QueryParams	HTTPQueryParamMatch ↗
Method	HTTPMethod ↗

Filters

Определяет преобразования или модификации, применяемые к запросам/ответам.

Типы фильтров

Тип	Метод	Типы значений	Описание
RequestHeaderModifier	`Set`	имя (строка) + значение (строка)	Перезаписывае заголовок запроса с указанным именем и значением
	`Add`	имя (строка) + значение	Добавляет заголовок к

Тип	Метод	Типы значений	Описание
		(строка)	запросу, дописывая к существующим значениям
	`Remove`	[]string	Удаляет указанные заголовки из запроса (без учёта регистра)
ResponseHeaderModifier	`Set`	имя (строка) + значение (строка)	Перезаписывае заголовок ответ с указанным именем и значением
	`Add`	имя (строка) + значение (строка)	Добавляет заголовок к ответу, дописывая к существующим значениям
	`Remove`	[]string	Удаляет указанные заголовки из ответа (без учё регистра)
RequestRedirect	`Scheme`	строка	Схема для заголовка Location (http/https)
	`Hostname`	PreciseHostname	Имя хоста для заголовка Location

Тип	Метод	Типы значений	Описание
	`ReplaceFullPath`	строка	Замена всего пути запроса
	`ReplacePrefixMatch`	строка	Замена совпадающего префикса пути
	`Port`	PortNumber	Порт для заголовка Location
	`StatusCode`	int	HTTP статус перенаправлен
URLRewrite	`Hostname`	PreciseHostname	Имя хоста для переписывания запросе
	`ReplaceFullPath`	строка	Замена всего пути запроса
	`ReplacePrefixMatch`	строка	Замена совпадающего префикса пути
CORS	`AllowOrigins`	[]string	Список разрешённых источников для CORS-запросов

Тип	Метод	Типы значений	Описание
	`AllowMethods`	[]HTTPMethod	Список разрешённых HTTP-методов
	`AllowHeaders`	[]string	Список разрешённых заголовков в CORS-запроса
	`ExposeHeaders`	[]string	Список заголовков, доступных клиенту в ответ
	`MaxAge`	Duration	Время кэширования д предварительно CORS-ответа
	`AllowCredentials`	bool	Разрешены ли учётные данны CORS-запроса

Примечания:

- `RequestRedirect` и `URLRewrite` не могут использоваться одновременно в одном правиле
- `ReplacePrefixMatch` совместим только с `PathPrefix` в HTTPRouteMatch
- Имена заголовков нечувствительны к регистру согласно RFC 7230
- Несколько значений одного заголовка должны использовать формат с разделением запятыми согласно RFC 7230

Ссылки на фильтры

Тип фильтра	Официальная документация
RequestHeaderModifier	HTTPHeaderFilter ↗
ResponseHeaderModifier	HTTPHeaderFilter ↗
RequestRedirect	HTTPRequestRedirectFilter ↗
URLRewrite	HTTPURLRewriteFilter ↗
CORS	HTTPCORSFilter ↗
RequestMirror	HTTPRequestMirrorFilter ↗
HTTPExternalAuthFilter	HTTPExternalAuthFilter ↗

Backend

Определяет целевой сервис(ы), на которые должны перенаправляться подходящие запросы.

Каждый сервис может иметь поле `weight` для указания доли трафика, направляемой на этот сервис.

Расширенные настройки правил HTTPRoute

Правила HTTPRoute поддерживают дополнительные параметры конфигурации, такие как политики повторных попыток, таймауты и другие параметры управления трафиком.

Таймауты

Поле	Спецификация
`.spec.rules[].timeouts`	HTTPRouteTimeouts ↗

Повторные попытки

Поле	Спецификация
`.spec.rules[].retry`	HTTPRouteRetry ↗

Сессионная аффинность

Настраивает параметры сессионной аффинности, чтобы обеспечить маршрутизацию запросов от одного клиента к одному и тому же бэкенду.

Поле	Спецификация
`.spec.rules[].sessionPersistence`	SessionPersistence ↗

Следующий шаг

Задачи для Envoy Gateway

Связанные задачи

- Как прикрепиться к слушателю, созданному в другом namespace

Настройка ALB

WARNING

ALB устарел. Пожалуйста, используйте вместо него ingress-nginx-operator или envoy-gateway.

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 создает соответствующие деплойменты и сервисы в ответ.

Для каждого ALB соответствующий Deployment отслеживает все Frontends и Rules, прикрепленные к этому ALB, и маршрутизирует запросы к бэкендам на основе этих

конфигураций.

Предварительные требования

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

Настройка ALB

Конфигурация ALB состоит из трех 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, связанная с ресурсами

Поля, связанные с ресурсами, описывают конфигурацию деплоймента для alb.

Поле	Тип	Описание
`.spec.config.nodeSelector`	map[string]string	селектор узлов для alb
`.spec.config.replicas`	int, необязательно, по умолчанию 3	количество реплик для 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 сети описывают, как получить доступ к ALB. Например, в режиме `host` alb использует hostnetwork, и вы можете получить доступ к ALB по IP узла.

Поле	Тип	Описание
`.spec.config.networkMode`	string: `host` или `container`, необязательно, по умолчанию `host`	В режиме `container` оператор создает LoadBalancer Service и использует его адрес как адрес ALB.
`.spec.address`	string, обязательно	вы можете вручную указать адрес alb

Поле	Тип	Описание
`.spec.config.vip.enableLbSvc`	bool, необязательно	Автоматически true в режиме `container`.
`.spec.config.vip.lbSvcAnnotations`	map[string]string, необязательно	Дополнительные аннотации для LoadBalancer Service.

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 разделу ingress-sync.
3. В веб-интерфейсе правила, созданные в проекте X, не будут видны и не могут быть настроены в проекте Y.

Если вы включите портовый проект и назначите диапазон портов проекту, это означает:

1. Вы не сможете создавать порты, не принадлежащие диапазону портов, назначенному проекту.

Дополнительная настройка

Существуют некоторые глобальные настройки, которые можно изменить в alb cr.

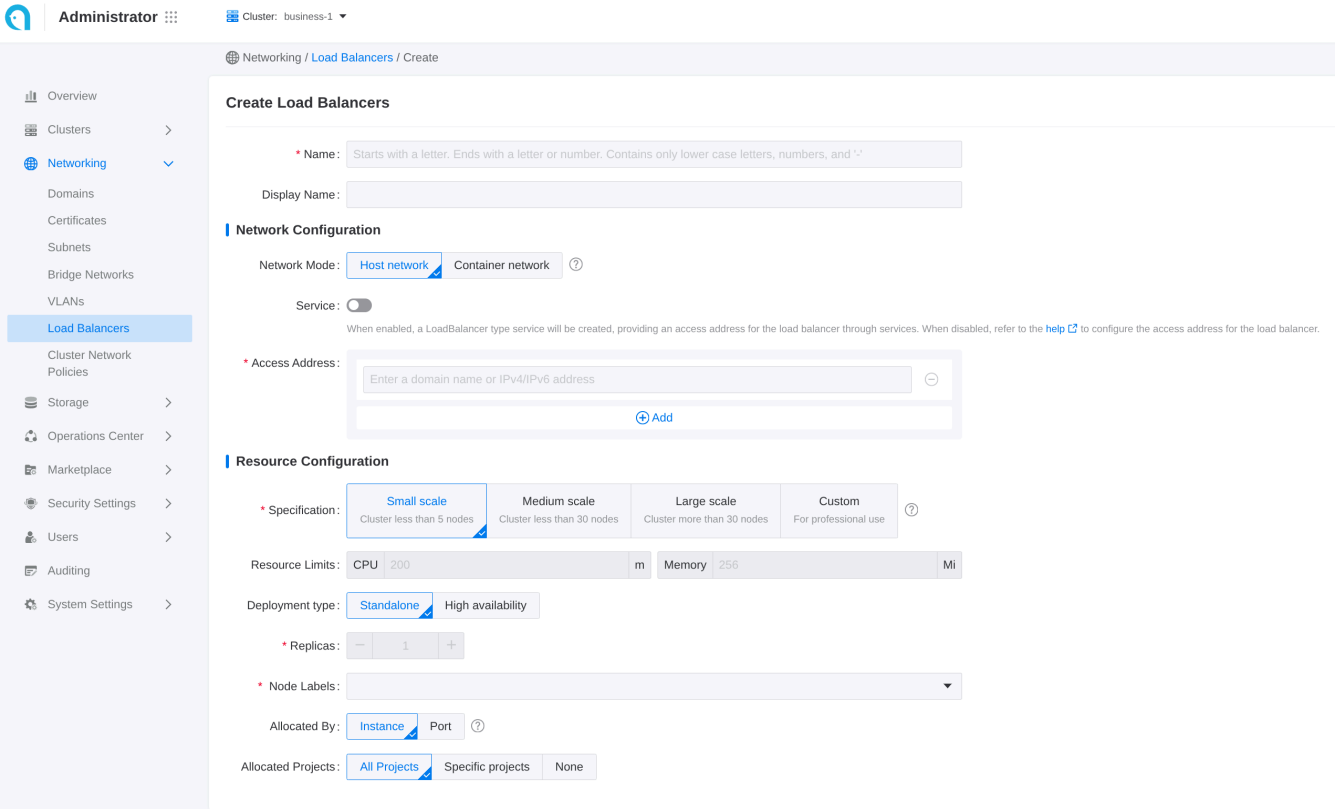
- Привязка NIC

- Синхронизация Ingress

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, и вы можете выбрать проект ниже, `port` — режим порт-проекта, после создания alb можно назначить диапазон портов

Через CLI

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

Обновление

Через веб-консоль

> **NOTE**
>
> Обновление балансировщика нагрузки приведет к прерыванию сервиса на 3-5 минут.
>
> Пожалуйста, выберите 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, разумно задавайте спецификации 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 состояние **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
```

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 используйте frontend для прямой настройки backend-сервиса.
- В L7 Proxy используйте frontend для настройки портов слушателя и rule для настройки backend-сервиса.

Если необходимо добавить HTTPS-порт слушателя, следует также обратиться к администратору для назначения TLS-сертификата текущему проекту для шифрования.

Предварительные требования

Сначала создайте ALB.

Настройка 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) Метка alb: обязательна, 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 этим портом.

- L7 протоколы https|http|grpcs|grpc и L4 протоколы tcp|udp.
- При выборе HTTPS необходимо добавить сертификат; добавление сертификата для протокола gRPC необязательно.
- При выборе протокола gRPC протокол бэкенда по умолчанию gRPC, который не поддерживает сессионную привязку. Если для gRPC протокола установлен сертификат, балансировщик нагрузки снимет сертификат gRPC и перенаправит незашифрованный трафик gRPC на backend-сервис.
- При использовании кластера Google GKE балансировщик нагрузки с одинаковым **типом контейнерной сети** не может одновременно иметь протоколы слушателя TCP и UDP.

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

6 backendProtocol: протокол, используемый backend-сервисом.

7 По умолчанию `serviceGroup`:

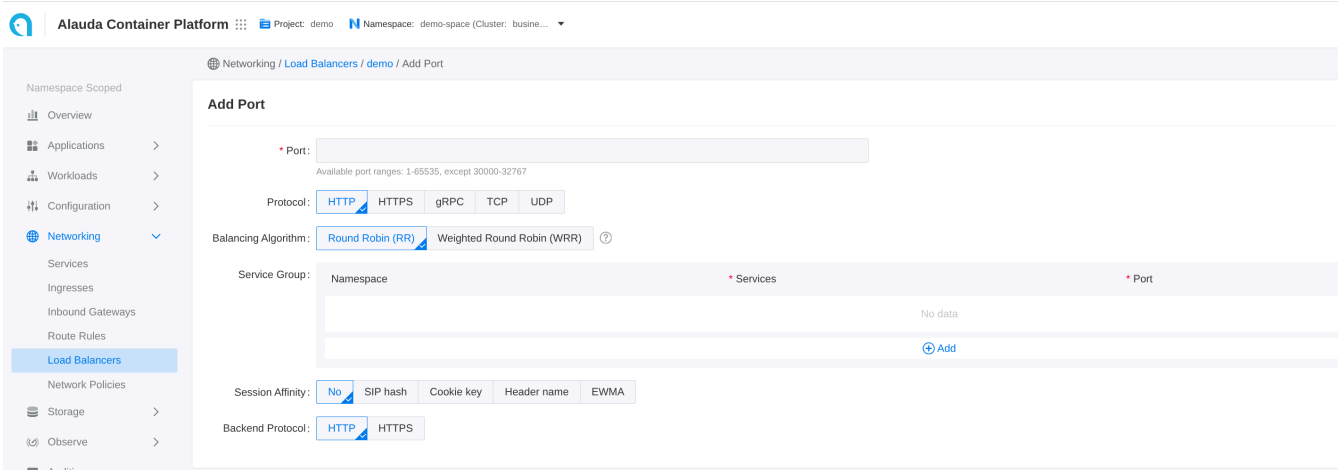
- L4 proxy: обязательно. ALB напрямую перенаправляет трафик в группу сервисов по умолчанию.
- L7 proxy: необязательно. ALB сначала сопоставляет правила (Rules) на этом Frontend; если совпадений нет, используется группа сервисов по умолчанию.

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, вы можете задать более разнообразные правила сопоставления backend-сервисов rules. Балансировщик нагрузки сначала сопоставит соответствующий backend-сервис согласно заданным правилам; если совпадение не найдено, он перейдет к backend-сервисам, соответствующим вышеупомянутой 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**, удалять связанные порты слушателей могут только администраторы в режиме **Administrator**.

Rule

Rule — это кастомный ресурс (CR), который определяет, как входящие запросы сопоставляются и обрабатываются ALB.

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

Предварительные требования

- Настройка ALB
- Настройка Frontend

Ниже приведен демонстрационный rule для быстрого ознакомления с использованием правил.

> **NOTE**
>
> rule должен быть прикреплен 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, приведенное ниже правило 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 заканчивается на *.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. Для настройки приоритета 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 совпал с правилом, к запросу можно применить следующие действия

Функция	Описание	Ссылка
Timeout	Настройка таймаутов 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 запросов или ответов.	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 распределенного трассирования и мониторинга.	otel
Keepalive	Включение или отключение функции keepalive для приложения.	keepalive

Backend

backend protocol

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

Группа сервисов и политика сессионной привязки

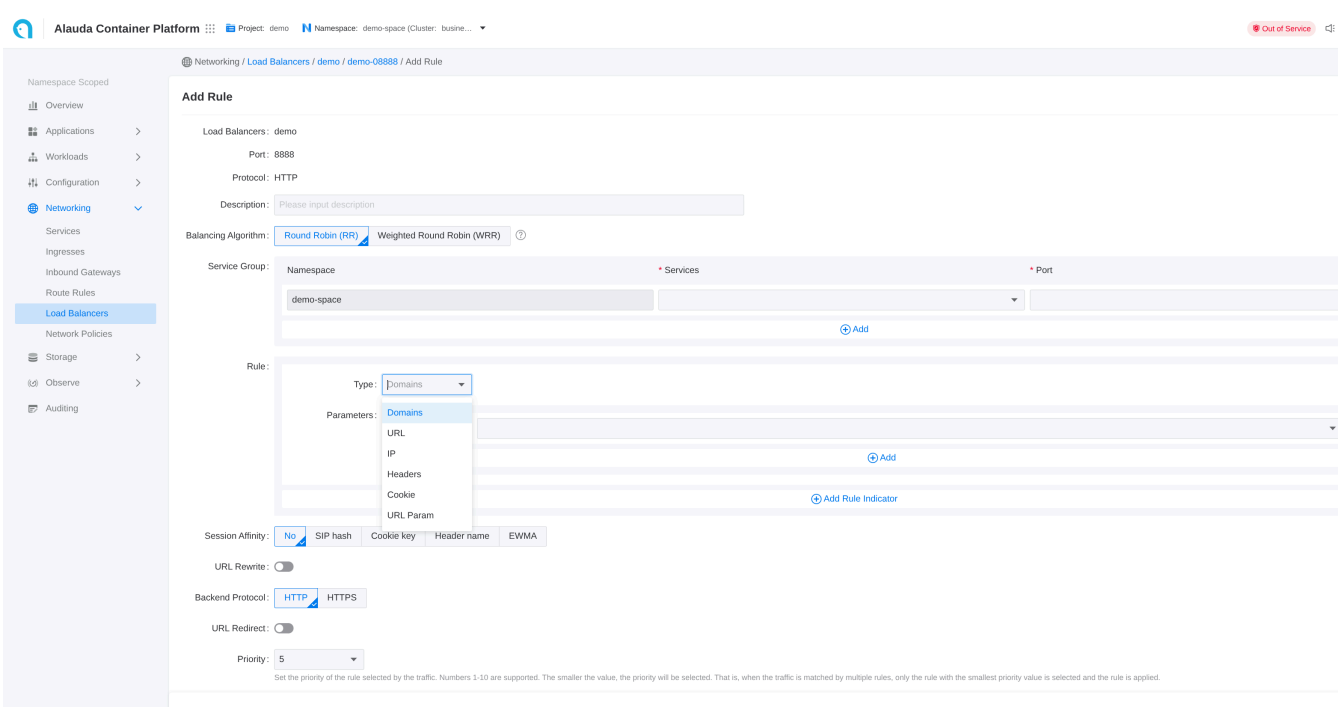
В правиле можно настроить один или несколько сервисов.

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

Подробнее смотрите в разделе Балансировка нагрузки.

Операции с Rule

Через веб-консоль



1. Перейдите в **Container Platform**.
2. В левой навигационной панели нажмите **Network** > **Load Balancing**.
3. Нажмите на имя балансировщика нагрузки.
4. Выберите имя порта слушателя.

5. Нажмите **Add Rule**.
6. Настройте соответствующие параметры согласно описаниям ниже.
7. Нажмите **Add**.

Каждое поле ввода в веб-интерфейсе соответствует полю CR

Через CLI

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

HTTPS

Если протокол frontend (ft) — HTTPS или GRPCS, правило также может быть настроено для использования HTTPS.

Вы можете указать сертификат либо в правиле, либо в ingress, чтобы сопоставить сертификат для конкретного порта.

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

Аннотация сертификата в Ingress

Сертификаты могут ссылаться на ресурсы в других пространствах имен через аннотацию.

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

Сертификат в Rule

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 frontend с протоколом https
2. создайте rule с backend-протоколом 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 frontend с протоколом https
2. создайте rule с backend-протоколом https и укажите сертификат через `.spec.certificate_name`

Ingress

Синхронизация Ingress

Каждый ALB создает IngressClass с таким же именем и обрабатывает ingress в рамках одного проекта.

Если в пространстве имен ingress есть метка `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 создает `Frontend` или Rule. Поле `source` определяется следующим образом:
>
> 1. `spec.source.type` в настоящее время поддерживает только `ingress`.
> 2. `spec.source.name` — имя ingress.

3. `spec.source.namespace` — пространство имен ingress.

SSL стратегия

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

Вы можете настроить кастомный ресурс ALB следующими параметрами:

- `.spec.config.defaultSSLStrategy`: определяет SSL стратегию для ingress без сертификатов
- `.spec.config.defaultSSLCert`: задает 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

Логи и мониторинг

Комбинируя 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 балансировщика нагрузки***.
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 балансировщика нагрузки***.
4. Во вкладке **Monitoring** просмотрите тенденции 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.

Настройка 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 при DNS-запросах 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 этого 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 вновь созданных 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 настроена NetworkPolicy, необходимо дополнительно разрешить трафик в обоих направлениях для node CIDR и nodeLocalDNSIP в networkPolicy, чтобы обеспечить корректную связь.

WARNING

Примечания к обновлению 4.2.x

При обновлении плагина с версий ниже 4.2.0 (исключая 4.2.0) до 4.2.x требуется выполнить следующие шаги из-за проблем совместимости ResourcePatch:

Перед обновлением:

- Зафиксируйте значение параметра `--node-local-dns-ip` из конфигурации ResourcePatch kube-ovn-controller
- Удалите ResourcePatch для ресурса `deploy/kube-ovn-controller`

После обновления:

- Вручную добавьте зафиксированный параметр `--node-local-dns-ip` обратно в конфигурацию kube-ovn-controller

Примечание: Эта проблема совместимости решена в версии 4.3 и выше, поэтому при обновлении на 4.3+ ручные действия не требуются.

Installation

Install via Marketplace

1. Перейдите в **Administrator** > **Marketplace** > **Cluster Plugins**.
2. В списке плагинов найдите "**Alauda Build of NodeLocal DNSCache**".
3. Нажмите **Install**, чтобы открыть страницу настройки установки.
4. Настройте необходимые параметры:

Parameter	Description	Example Value
IP	IP-адрес локального DNS-кэша узла. 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 примечаниями 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 включена NetworkPolicy, необходимо настроить соответствующие правила, разрешающие DNS-трафик 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
```

Configure CoreDNS

Содержание

Overview

CoreDNS — это служба DNS по умолчанию для кластеров Kubernetes. В этом руководстве описывается, как настроить псевдонимы хостов, селекторы узлов и толерантности для CoreDNS.

Configuration

1. Перейдите в **Administrator** > **Marketplace** > **Cluster Plugins**.
2. Найдите "**Alauda Build of CoreDNS**" и нажмите **Update**.
3. Настройте следующие параметры:

Host Alias

Настройка пользовательских записей разрешения DNS.

Parameter	Description
IP	IP-адрес для разрешения
Domains	Доменные имена (разделённые пробелами) **Пример**: `example.com test.example.com`

Node Selectors

Укажите, на каких узлах должны запускаться поды CoreDNS.

Parameter	Description
Label Key	Ключ метки для сопоставления с узлами
Label Value	Значение метки для сопоставления с узлами

Node Tolerations

Разрешить запуск подов CoreDNS на узлах с taints.

Parameter	Description
Key	Ключ taint для толерантности
Value	Значение taint (необязательно)
Type	`NoSchedule`, `PreferNoSchedule` или `NoExecute`

4. Нажмите **Update** для применения конфигурации.

Как сделать

Задачи для Ingress-Nginx

Задачи для Ingress-Nginx

Задачи для Envoy Gateway

Задачи для Envoy Gateway

Предварительные требования

Введение

Общие задачи для конфигурации Route

Расширенная конфигурация

Дополнительная конфигурация

Soft Data Center LB Solution (Alpha)

Soft Data Center LB Solution (Alpha)

Предварительные требования

Процедура

Проверка

Kube OVN

Понимание Kube-OVN CNI

Компоненты Upstream OVN/OVS

Основной Controller и Agent

Инструменты мониторинга, эксплуатаци

Подготовка физической сети

Инструкция по использованию

Объяснение терминологии

Требования к среде

Пример конфигурации

Автоматич

Процедура

Изоляция меж

Межкластерное соединение (Alpha)

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

Предварительные требования

Построен контроллер подключения многозвенных Kube-OVN

Развертывание контроллера межкластерного соединения в глобальном кластере

Присоединение к межкластерному соединению

Соответствующие операции

Настройка

Описание Egre

Предварительны

Использование

Параметры ко

Примечания

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

Установка Multus CNI

Создание подсетей

Создание Pod с несколькими сетевыми

Проверка создания двух сетевых интерс

Дополнительные возможности

Настройка

Инструкции

Меры предосто

Настройка

Поведение MT

Настройка пар

Настройка Endpoint Health Checker

Настройка Endpoint Health Checker

Overview

Key Features

Installation

How It Works

How To Activate

Uninstallation

alb

Задачи для ALB

Как задать NodeSelector и Tolerations для alb-operator

Как задать NodeSelector и Tolerations для alb

Задачи для Ingress-Nginx

Содержание

Как настроить

Предварительные требования

Установка ingress-nginx

Максимальное количество соединений

Max-Worker-Connections ↗

Таймаут запроса

Настройка таймаута запроса ↗

Сессионная аффинность (Sticky Sessions)

Настройка sticky sessions ↗

Модификация заголовков

действие	ссылка
установить заголовок в запросе	proxy-set-header ↗
удалить заголовок в запросе	установить пустой заголовок в запросе
установить заголовок в ответе	configuration-snippets ↗ с директивой more-set-header ↗
удалить заголовок в ответе	hide-headers ↗

Перезапись URL

rewrite ↗

HSTS (HTTP Strict Transport Security)

настройка HSTS ↗

Ограничение скорости

настройка ограничения скорости ↗

WAF

modsecurity ↗

Управление заголовком Forward

x-forwarded-prefix-header ↗

HTTPS

TLS повторное шифрование и проверка сертификата бэкенда

проверка сертификата backend https ↗

TLS терминация на краю

backend protocol ↗

Passthrough

ssl-passthrough ↗

Сертификат по умолчанию

default-ssl-certificate ↗

Добавление аннотации Pod в IngressNginx

Добавить аннотацию pod

Сохранение исходного IP

При прохождении трафика через балансировщики нагрузки или прокси, исходный IP-адрес клиента может быть утерян из-за NAT (Network Address Translation). Сохранение исходного IP важно для:

- Контроля доступа и политик безопасности
- Точного логирования и аналитики
- Ограничения скорости на клиента
- Маршрутизации на основе геолокации

Через HAProxy Proxy Protocol

Как это работает

PROXY protocol ↗ — это сетевой протокол для сохранения информации о клиентском соединении при проксировании TCP-соединений. Он работает путем добавления заголовка к TCP-соединению, содержащего исходный IP и порт клиента.

Поток трафика:

1. Клиент подключается к балансировщику HAProxy
2. HAProxy добавляет заголовок PROXY protocol с исходным IP клиента к соединению
3. Ingress-Nginx принимает соединение и разбирает заголовок PROXY protocol
4. Ingress-Nginx извлекает реальный IP клиента из заголовка
5. Бэкенд-приложения получают корректный IP клиента в заголовках `X-Forwarded-For` и `X-Real-IP`

Преимущества:

- Работает с любым балансировщиком, поддерживающим PROXY protocol (HAProxy, AWS NLB и др.)
- Сохраняет исходный IP через несколько уровней прокси
- Не влияет на маршрутизацию или выбор узла

Особенности:

- Балансировщик и Ingress-Nginx должны быть настроены на использование PROXY protocol
- Весь трафик к Ingress-Nginx должен использовать PROXY protocol после включения (смешивание PROXY и обычного трафика приведет к сбоям соединения)

Как настроить

Настройте HAProxy для отправки заголовков PROXY protocol, затем разверните ingress-nginx с поддержкой proxy-protocol:

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

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

frontend tcp_front_443
    bind *:443
    mode tcp
    default_backend ingress_tcp_443

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

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

Для подробностей смотрите документацию PROXY protocol↗.

Примечание: HAProxy может использовать TCP режим для передачи трафика без обработки TLS-сертификатов. Поскольку PROXY protocol работает на TCP-уровне, вы можете позволить Ingress-Nginx самостоятельно обрабатывать HTTPS-терминацию и управление сертификатами, исключая необходимость настройки сертификатов в HAProxy.

Через MetalLB с externalTrafficPolicy=Local

Как это работает

При использовании Kubernetes Service с `type: LoadBalancer` поведение по умолчанию (`externalTrafficPolicy: Cluster`) выполняет source NAT, заменяя IP клиента на IP узла. Установка `externalTrafficPolicy: Local` сохраняет исходный IP за счет:

1. **Прямой маршрутизации**: трафик направляется только на поды, расположенные на том же узле, который получил трафик

2. **Отсутствия SNAT**: kube-proxy не выполняет source NAT, сохраняя исходный IP клиента
3. **Проверок здоровья**: в пул балансировщика включаются только узлы с работоспособными локальными подами

Поток трафика:

1. Клиент подключается к виртуальному IP MetalLB
2. MetalLB направляет трафик напрямую на узел с подами Ingress-Nginx
3. Трафик идет напрямую к локальному поду Ingress-Nginx без SNAT
4. Ingress-Nginx видит реальный IP клиента
5. Бэкенд-приложения получают корректный IP клиента в заголовках

Преимущества:

- Простая настройка, не требует дополнительных протоколов
- Встроенная функция Kubernetes
- Меньшая задержка (нет дополнительного прокси-прыжка)

Особенности:

- **Неравномерное распределение нагрузки**: трафик может идти только на узлы с локальными подами, что может привести к дисбалансу нагрузки
- **Планирование подов**: поды Ingress-Nginx должны быть запланированы на узлах, доступных MetalLB (используйте nodeSelector для согласования)
- **Поведение проверок здоровья**: если все локальные поды неработоспособны, узел полностью исключается из балансировки

Как настроить

Разверните ingress-nginx с `externalTrafficPolicy: Local` и убедитесь, что размещение подов соответствует конфигурации MetalLB:

```yaml
apiVersion: ingress-nginx.alauda.io/v1
kind: IngressNginx
metadata:
  name: demo
  namespace: ingress-nginx-operator
spec:
  controller:
    service:
      type: LoadBalancer                    # Использовать MetalLB для со
здания сервиса LoadBalancer
      externalTrafficPolicy: Local          # Сохранять исходный IP, напр
авляя трафик только на локальные поды
      annotations:
        metallb.universe.tf/address-pool: demo-pool  # Указать пул IP-адр
есов MetalLB для использования
    nodeSelector:                           # Планировать поды только на
узлах с этими метками. Этот селектор должен совпадать с nodeSelector пула
адресов MetalLB
      ingress-nginx: "true"
```

Важно: `nodeSelector` должен совпадать с узлами в конфигурации пула адресов MetalLB, чтобы гарантировать, что поды Ingress-Nginx размещаются на узлах, доступных MetalLB для маршрутизации трафика.

Для подробностей смотрите документацию externalTrafficPolicy↗.

Задачи для Envoy Gateway

Содержание

Предварительные требования

1. Настройка EnvoyGatewayCtl
2. Настройка Gateway
3. Настройка Route

Введение

При применении изменений конфигурации в Gateway API доступны три основных подхода:

1. Прямое изменение HTTP/TCP/UDP Route или `Gateway`.
2. Изменение через PolicyAttachment, предоставляемое `Gateway Api` и `Envoy Gateway`.
3. Изменение на уровне глобальной конфигурации экземпляра `envoy-gateway`.

HTTP/TCP/UDP Route

- Спецификация HTTPRoute ↗
- Спецификация TCPRoute ↗
- Спецификация UDPRoute ↗

PolicyAttachment через TargetRefs

Envoy Gateway предоставляет расширенный механизм пользовательских политик, которые могут быть прикреплены к ресурсам gateway через модель PolicyAttachment Gateway API ↗.

Политики Envoy Gateway делятся на несколько типов, включая политики безопасности, управления трафиком и другие. Эти политики могут применяться к разным уровням ресурсов, таким как Gateway, HTTPRoute или Service.

Механизм PolicyAttachment Gateway API позволяет пользователям декларативно прикреплять политики к ресурсам gateway. Этот механизм реализован через поле `targetRefs`, которое указывает целевой ресурс для применения политики. Например, политики могут быть прикреплены к конкретным Gateway, HTTPRoute или Service.

Типы политик, поддерживаемые Envoy Gateway, включают:

Тип политики	Описание
ClientTrafficPolicy ↗	Конфигурация, связанная с путем коммуникации клиент-прокси, включая параметры такие как таймауты, повторные попытки, настройки keepalive и др.
BackendTrafficPolicy ↗	Конфигурация, связанная с путем коммуникации прокси-бэкенд, включая параметры такие как таймауты, повторные попытки, настройки keepalive и др.
SecurityPolicy ↗	Конфигурация, связанная с механизмами и контролями безопасности, такими как аутентификация и авторизация.

Используя механизм PolicyAttachment, пользователи могут гибко добавлять, изменять или удалять политики без изменения основных определений ресурсов, достигая разделения ответственности и лучшего управления ресурсами.

Глобальная конфигурация

Конфигурация, связанная с самим экземпляром `envoy-gateway` или глобальная конфигурация, относящаяся ко всем gateway этого экземпляра, например режим развертывания или маршрутизация бэкенда, относится к этой категории.

Рекомендуется использовать EnvoyGatewayCtl для управления такими глобальными конфигурациями.

Общие задачи для конфигурации Route

Функция	CR	Описание
Auth	envoygateway:SecurityPolicy	Авторизация ↗
CORS	gatewayapi:HTTPRoute	Cross-Origin Resource Sharing ↗
Модификация заголовков	gatewayapi:HTTPRoute	Модификация HTTP заголовков ↗
HTTP Redirect	gatewayapi:HTTPRoute	HTTP Redirect ↗
L7 Timeout	gatewayapi:HTTPRoute	Таймауты запросов ↗
SessionAffinity	gatewayapi:HTTPRoute	Session Affinity/Sticky Sessions ↗
L7 Keepalive	envoygateway:ClientTrafficPolicy	Настройки L7 Keepalive Timeout ↗
L4 Keepalive	envoygateway:ClientTrafficPolicy	Настройки L4 TCP Keepalive ↗
UrlRewrite	gatewayapi:HTTPRoute	Перезапись URL ↗
Retry	gatewayapi:HTTPRoute или envoygateway:BackendTrafficPolicy	Конфигурация повторных попыток через HTTPRoute ↗ Конфигурация повторных попыток через EnvoyGateway ↗
GZip	envoygateway:BackendTrafficPolicy	HTTP сжатие ↗

Расширенная конфигурация

OpenTelemetry (Otel)

Пожалуйста, следуйте инструкциям в Интеграция OpenTelemetry↗, но используйте `EnvoyGatewayCtl` для изменения `envoy-gateway-config`.

Как прикрепиться к Listener, созданному в другом Namespace

В конфигурации listener Gateway необходимо указать, какие namespaces разрешено прикреплять к нему Routes.

```
apiVersion: gateway.networking.k8s.io/v1
kind: Gateway
metadata:
  name: example-gateway
spec:
  listeners:
    - name: http-80
      protocol: HTTP
      port: 80
      allowedRoutes:
        namespaces:
          from: All # без ограничений
    - name: http-81
      protocol: HTTP
      port: 81
      allowedRoutes:
        namespaces:
          from: Same # разрешены только маршруты из того же namespace
    - name: http-82
      protocol: HTTP
      port: 82
      allowedRoutes:
        namespaces:
          from: Selector
          selector:
            matchLabels:
              team: frontend # разрешены только маршруты из namespace с м
еткой team=frontend
```

Пожалуйста, обратитесь к Cross-Namespace routing ↗ для получения дополнительной информации.

Как использовать сертификат, созданный в другом Namespace

Для использования сертификата, созданного в другом namespace, необходимо создать `ReferenceGrant` в namespace, где создан сертификат. Пожалуйста, следуйте инструкциям в cross-namespace-certificate-references ↗ и referencegrant ↗.

> **NOTE**
>
> Нельзя указывать отдельные ресурсы `secret`; необходимо разрешить весь namespace

Как использовать SSL passthrough

Пожалуйста, следуйте инструкциям в

- tls ↗
- tls-passthrough ↗

Как изменить SSL Cipher

Пожалуйста, следуйте инструкциям в customize-gateway-tls-parameters ↗

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

`.spec.tls` в `ClientTrafficPolicy` соответствует clienttlssettings ↗

Как указать NodePort при использовании NodePort Service

При использовании NodePort сервиса Kubernetes назначает номер порта NodePort для каждого порта сервиса. При обращении к сервису через IP узла следует использовать соответствующий порт NodePort, а не порт сервиса.

Существует два подхода для решения этой задачи:

Ручное получение назначения NodePort, следуя инструкции get nodeport from svc port

Ручное указание NodePort в конфигурации `EnvoyProxy` вместо автоматического назначения Kubernetes.

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

1 Используйте поле patch для патча сгенерированного ресурса сервиса, чтобы указать NodePort

NOTE

NodePort может быть только в определённом диапазоне, обычно `30000-32767`. Если вы хотите, чтобы порт слушателя Gateway и NodePort совпадали, порт слушателя также должен находиться в диапазоне NodePort.

Как добавить аннотацию Pod в EnvoyGateway

Добавление аннотации Pod

Как задать NodeSelector и Tolerations для envoy-gateway-operator

Обновите ресурсы `Subscription`

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

Как задать NodeSelector и Tolerations для envoy-gateway

Обновите ресурсы `EnvoyGatewayCtl`

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

Как задать NodeSelector и Tolerations для envoy-proxy

Обновите ресурсы `EnvoyProxy`

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

Дополнительная конфигурация

Пожалуйста, обратитесь к EnvoyGateway Tasks ↗

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 конфигурация только для IPv4, только для IPv6 или для двойного стека IPv4 и IPv6.

Содержание

Предварительные требования

1. Подготовьте два или более узлов-хоста в качестве LB. Рекомендуется установить на узлы LB операционную систему Ubuntu 22.04 для сокращения времени перенаправления трафика LB на аномальные backend-узлы.
2. Предварительно установите следующее программное обеспечение на все узлы-хосты внешнего LB (в этом разделе в качестве примера рассматриваются два внешних узла LB):
   - `ipvsadm`
   - `container-runtime, например containerd`

3. Убедитесь, что `container-runtime` настроен на автозапуск при загрузке каждого хоста.
4. Убедитесь, что часы каждого узла-хоста синхронизированы.
5. Подготовьте образ Keepalived, используемый для запуска сервиса 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. Получить адрес репозитория образа и суффикс версии можно следующим образом. В этом 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` для получения параметра **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}'` для получения **version suffix**.

Процедура

Примечание: Следующие операции необходимо выполнить один раз на каждом внешнем узле LB, при этом `hostname` 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 конфигурационные параметры 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, необязательно
      interface: "eth0"
      check_interval: 1 # необязательно, по умолчанию 1: интервал выпол
нения скрипта проверки
      check_timeout: 3  # необязательно, по умолчанию 3: таймаут скрипт
а 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 Keepaliv
ed, в сгенерированном keepalived.conf все IP на интерфейсе будут удален
ы.
      kube_lock:
        kubecfgs: # Список kube-config, используемый kube-lock для посл
едовательной попытки выбора лидера в Keepalived
          - "/live/cfg/kubecfg/kubecfg01.conf"
          - "/live/cfg/kubecfg/kubecfg02.conf"
          - "/live/cfg/kubecfg/kubecfg03.conf"
    ipvs: # Конфигурация 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 порто
м реального сервера
          virtual_server_config: |
            delay_loop 10  # Интервал выполнения проверки здоровья реал
ьного 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 посл
едовательной попытки 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::19
2:168:129:100" ]
      ports:
        - port: 80
          virtual_server_config: |
            delay_loop 10
            lb_algo rr
            lb_kind NAT
            protocol TCP
          raw_check: |
            TCP_CHECK {
                connect_timeout 1
                connect_port 1936
            }
```

6. 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-certifi
cate-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 узлах LB, присвоив им имена с индексом, например, `kubecfg01.conf`, и измените адреса узлов `apiserver` в этих трех файлах 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 сертификатов.

```
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 узел LB и запустите Keepalived с помощью nerdctl.

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

10. 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, применяемые к ALB бизнес-кластера.

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

2. Выключите узел LB, 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. Используйте команду `curl` на узле, не являющемся LB, чтобы проверить нормальность связи 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="htt
p://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at <a href="http://nginx.com/">nginx.co
m</a>.</p>

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

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

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

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

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

Kube OVN

Понимание Kube-OVN CNI

- Компоненты Upstream OVN/OVS
- Основной Controller и Agent
- Инструменты мониторинга, эксплуатаци

Подготовка физической сети

- Инструкция по использованию
- Объяснение терминологии
- Требования к среде
- Пример конфигурации

Автоматич

- Процедура
- Изоляция межд

Межкластерное соединение (Alpha)

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

- Предварительные требования
- Построен контроллер подключения многозвенных Kube-OVN
- Развертывание контроллера межкластерного соединения в глобальном кластере
- Присоединение к межкластерному соединению
- Соответствующие операции

Настройка

- Описание Egre
- Предварительны
- Использование
- Параметры ко
- Примечания

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

- Установка Multus CNI
- Создание подсетей
- Создание Pod с несколькими сетевыми
- Проверка создания двух сетевых интерс
- Дополнительные возможности

Настройка

- Инструкции
- Меры предосто

Настройка

Понимание Kube-OVN CNI

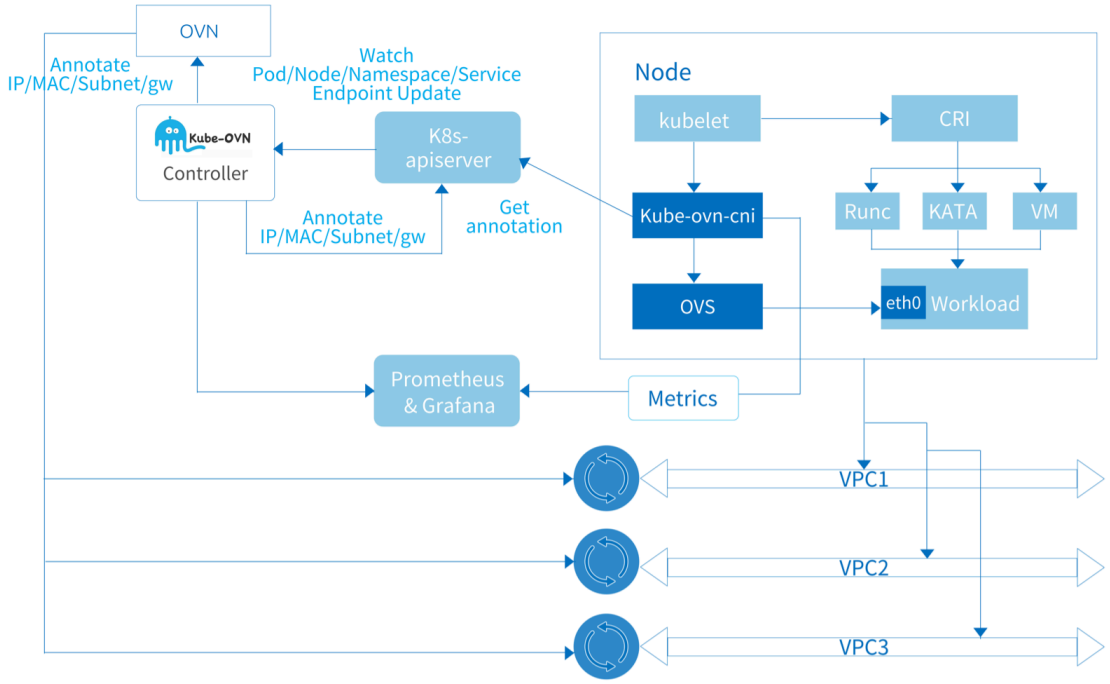
В этом документе описывается общая архитектура Kube-OVN, функциональность каждого компонента и их взаимодействие.

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 из области SDN в облачную нативную среду, таких как логические коммутаторы, логические маршрутизаторы, VPC, шлюзы, QoS, ACL и зеркалирование трафика.

Kube-OVN также сохраняет хорошую открытость для интеграции со многими технологическими решениями, такими как Cilium, Submariner, Prometheus, KubeVirt и др.

Компоненты Kube-OVN можно условно разделить на три категории.

- Компоненты Upstream OVN/OVS.
- Основной Controller и Agent.
- Инструменты мониторинга, эксплуатации и расширения.



Содержание

Компоненты Upstream OVN/OVS

Этот тип компонентов происходит из сообщества OVN/OVS с конкретными модификациями под сценарии использования Kube-OVN. OVN/OVS сам по себе является зрелой SDN-системой 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`, переводя таблицы логических потоков в реальные сетевые конфигурации.

Основной Controller и Agent

Эта часть является основным компонентом Kube-OVN, служащим 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 ведётся слежение, включают:

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

В качестве примера события 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. Затем `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.

Этот 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 дальнейшей обработки. 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:
    1. Создание или удаление пары veth и привязка или отвязка к портам OVS.
    2. Настройка портов OVS.
    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 внешний интерфейс для расширения основных сетевых возможностей Kube-OVN и упрощения ежедневных операций и обслуживания.

kube-ovn-speaker

Этот компонент — DaemonSet, работающий на специально помеченных узлах, который публикует маршруты во внешний мир, позволяя внешнему доступу к контейнерам напрямую по IP Pod.

kube-ovn-pinger

Этот компонент — DaemonSet, работающий на каждом узле для сбора информации о состоянии OVS, качестве сети узла, задержках сети и т.д.

kube-ovn-monitor

Этот компонент собирает информацию о состоянии OVN и метрики мониторинга.

kubectl-ko

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

Подготовка физической сети Kube-OVN Underlay

Сетевая инфраструктура контейнеров в режиме транспортного уровня Kube-OVN Underlay зависит от поддержки физической сети. Перед развертыванием сети Kube-OVN Underlay необходимо совместно с сетевым администратором спланировать и заранее выполнить соответствующие настройки физической сети для обеспечения сетевой связности.

Содержание

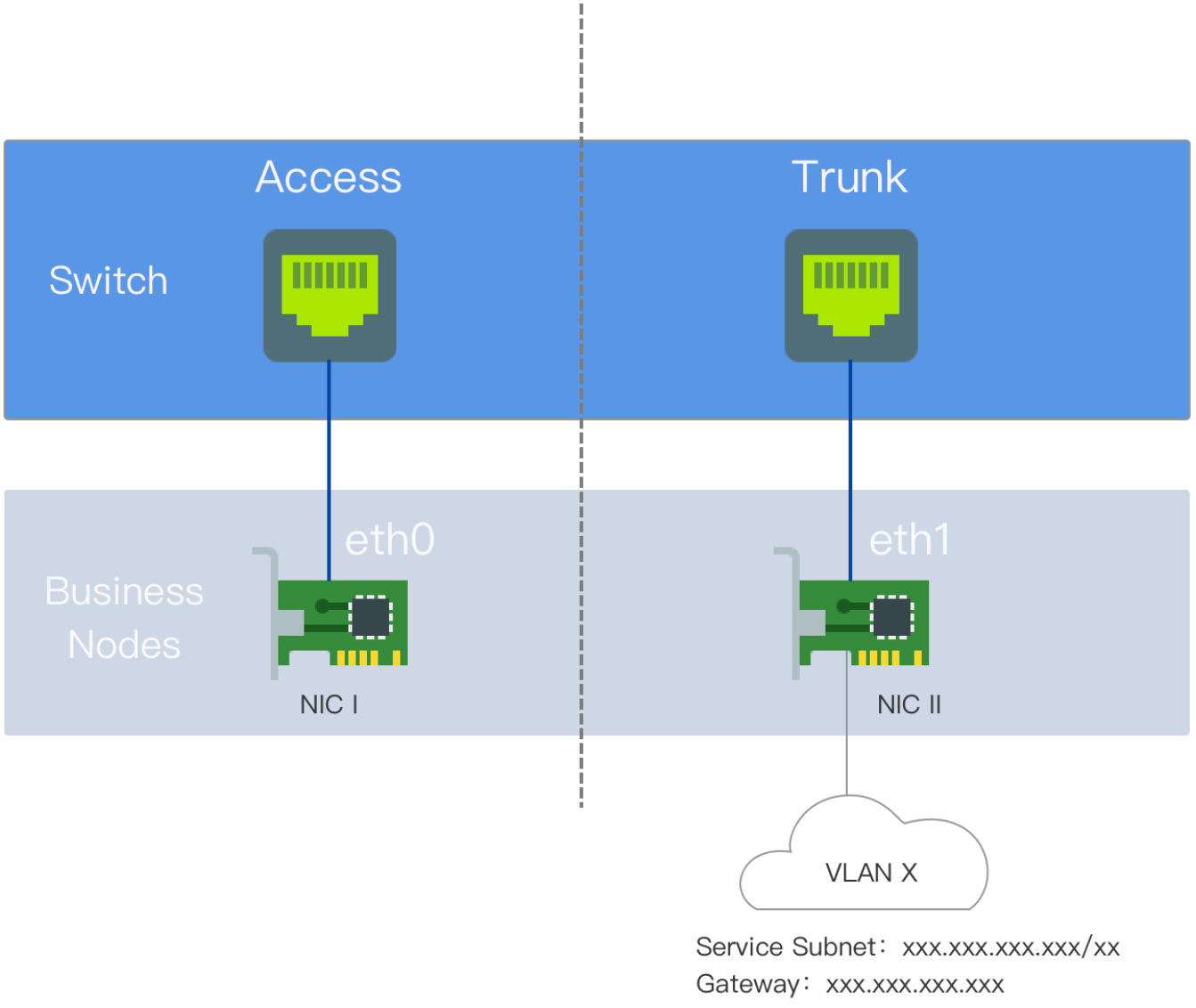
Инструкция по использованию

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, а должны работать через другие NIC.

Перед использованием убедитесь, что на сервере узла имеется как минимум **двухсетевой NIC**, при этом 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 настройки 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 компьютерных рабочих станций с похожими требованиями и обладает теми же свойствами, что и физически сформированная LAN. Поскольку VLAN разделяются логически, а не физически, рабочие станции в одном VLAN не ограничены одной физической зоной и могут находиться в разных физических сегментах LAN.

Основные преимущества VLAN включают:

- Сегментация портов. Даже на одном коммутаторе порты, принадлежащие разным VLAN, не могут взаимодействовать друг с другом. Физический коммутатор может функционировать как несколько логических коммутаторов. Это часто используется для контроля взаимного доступа между различными 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 мостит физический NIC в OVS и отправляет пакеты напрямую во внешнюю сеть через этот физический NIC. Возможности L2/L3 маршрутизации зависят от базовых сетевых устройств. Соответствующие шлюзы, 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 уметь пересылать пакеты обратно в локальную подсеть.
- Если на коммутаторе или мосту включена функция Hairpin, **Hairpin должен быть отключён**. В среде виртуальных машин VMware необходимо установить параметр **Net.ReversePathFwdCheckPromisc** на хосте VMware в значение **1**, тогда отключать Hairpin не требуется.
- Мостовой NIC **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 уровня IaaS (виртуализации)**
  - В средах 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**.
  - В публичных облаках, таких как AWS, GCE и Alibaba Cloud, сети в режиме Underlay не поддерживаются из-за отсутствия возможности задавать MAC-адреса пользователем.

Пример конфигурации

В данном примере узлы — это физические машины с двумя NIC. NIC Один — это NIC с маршрутом по умолчанию; NIC Два — NIC без маршрута по умолчанию и без настройки 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, позволяющем пропускать соответствующие 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 с 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  // Разрешить прохождение соответствующего VL
AN
#
```

Проверка сетевой связности

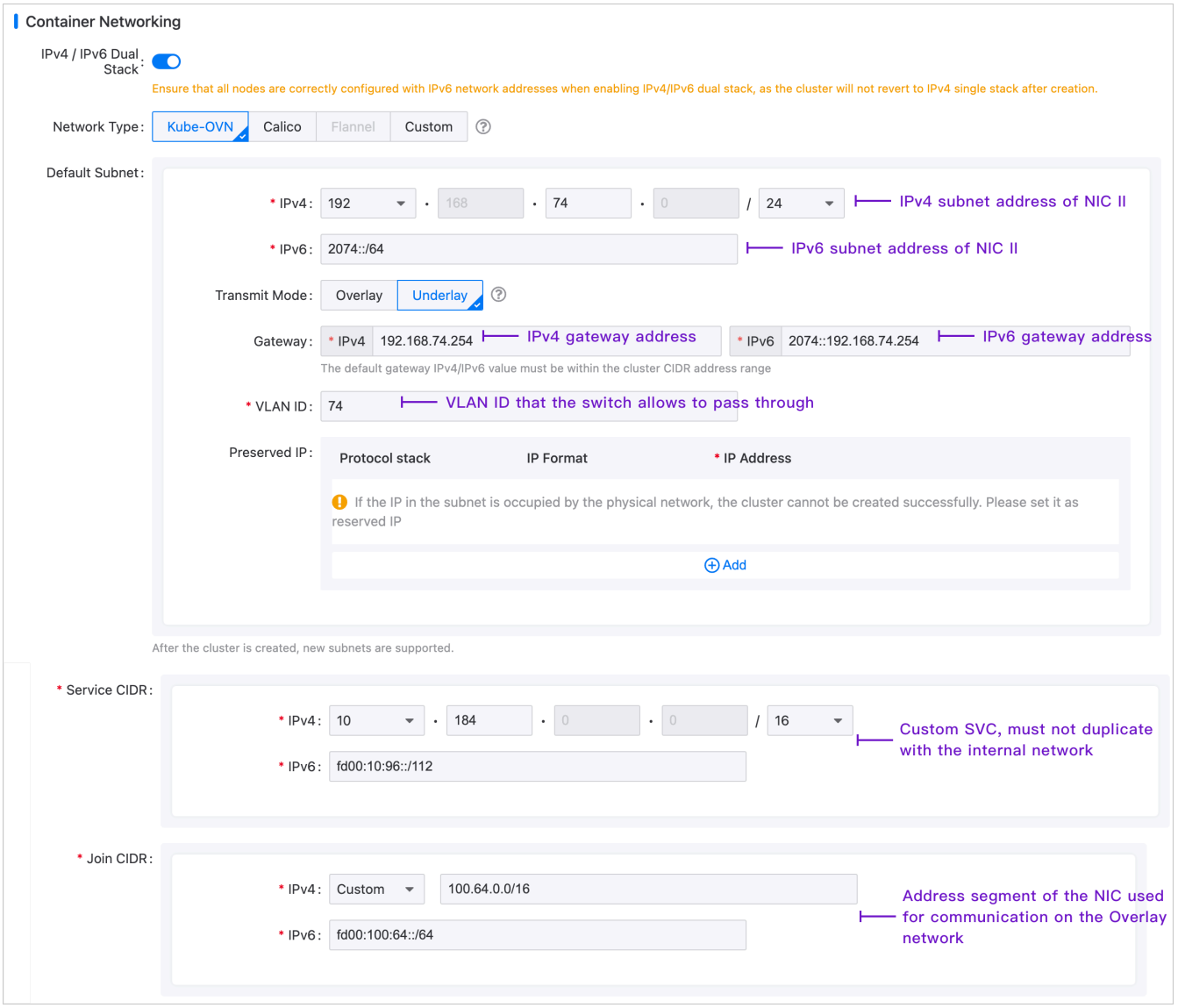
Проверьте, может ли NIC Два связаться с адресом шлюза:

```
ip link add ens224.74 link ens224 type vlan id 74  // Имя NIC — ens224, V
LAN ID — 74
ip link set ens224.74 up
ip addr add 192.168.74.200/24 dev ens224.74  // Выберите тестовый адрес в
подсети Underlay, здесь 192.168.74.200/24
ping 192.168.74.254  // Если пинг проходит, значит физическая среда соотв
етствует требованиям развертывания
ip addr del 192.168.74.200/24 dev ens224.74  // Удалите тестовый адрес по
сле 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 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

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 ⋮ > **Update** рядом с подсетью Underlay, которую необходимо изменить.
5. Измените YAML-файл, добавив поле `u2oInterconnection: true` в `Spec`.
6. Нажмите **Update**.

Примечание: Для вступления изменений в силу необходимо пересоздать существующие вычислительные компоненты в подсети Underlay.

Изоляция между подсетями Underlay с включённым u2oInterconnection

Когда у нескольких подсетей Underlay включён параметр `u2oInterconnection: true`, трафик между ними больше не проходит через физический шлюз, а маршрутизируется напрямую через внутреннюю сеть OVN.

Если необходимо изолировать две подсети Underlay при включённом `u2oInterconnection`, сначала нужно настроить параметр kube-ovn-controller, а затем настроить изоляцию подсетей.

Шаг 1: Настройка kube-ovn-controller

Измените Deployment kube-ovn-controller, чтобы отключить пропуск connection tracking для IP-адресов логических портов назначения:

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

Добавьте или измените следующий аргумент:

```
spec:
  template:
    spec:
      containers:
      - name: kube-ovn-controller
        args:
        - --ls-ct-skip-dst-lport-ips=false
```

CAUTION

`--ls-ct-skip-dst-lport-ips` управляет пропуском connection tracking (conntrack) для трафика, направленного на IP-адреса логических портов. Значение по умолчанию — `true`, что пропускает conntrack для повышения производительности. Установка в `false` не влияет на функциональность, но может немного снизить производительность.

Однако для подсетей Underlay с изоляцией на основе ACL **обязательно** установить значение `false`. В противном случае трафик от шлюза к Pod будет некорректно работать (например, ping-запросы доходят до Pod, но ответы отбрасываются), так как изоляция ACL использует `allow-related`, требующий состояния conntrack; без него ответы не распознаются как «связанные» и отбрасываются.

Шаг 2: Настройка изоляции подсети

Настройте подсеть с такими параметрами:

```
spec:
  u2oInterconnection: true
  acls:
  - action: drop
    direction: to-lport  # Направление входящего трафика (ingress)
    match: ip4.src == 172.20.0.0/16
    priority: 1002
  - action: drop
    direction: to-lport  # Направление входящего трафика
    match: ip4.src == 192.50.0.0/16
    priority: 1002
```

Параметры ACL:

Параметр	Описание
`action`	Действие: `allow`, `drop` или `allow-related`
`direction`	Направление трафика: `to-lport` (входящий) или `from-lport` (исходящий)
`match`	OVN-выражение для сопоставления с использованием полей L2-L4 и булевых операторов
`priority`	Приоритет правила (чем выше значение, тем выше приоритет; рекомендуемый диапазон: 1002-1899)

NOTE

- Поле `acls` обеспечивает приоритетную оценку правил, предоставляя большую гибкость по сравнению со стандартными Kubernetes NetworkPolicy.
- При использовании направления `to-lport` `ip4.src` означает исходный IP-адрес входящего трафика.
- **Рекомендуемый диапазон приоритетов**: от `1002` до `1899`, чтобы избежать конфликтов с системными правилами ACL.

Конфигурация сервиса LoadBalancer Kube-OVN Underlay + MetalLB

Содержание

Overview

Это решение предназначено для интеграции режима L2 MetalLB с сетевой подсистемой Kube-OVN Underlay. Оно позволяет использовать IP-адреса подсети Underlay в качестве VIP сервисов LoadBalancer MetalLB с прямой переадресацией трафика на backend Pods.

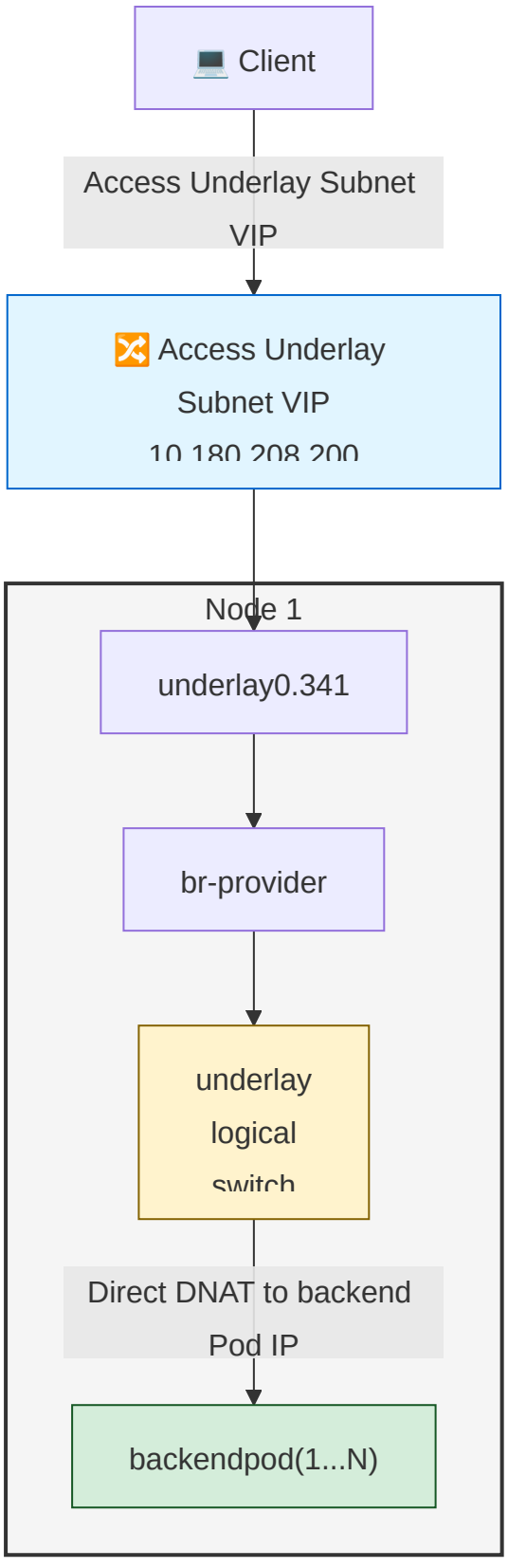
Prerequisites

Требования к окружению

- **Версия ACP**: >= 4.2.2
- **Поддержка IP**: только IPv4 (IPv6 в настоящее время не поддерживается)

Поток трафика

Диаграмма трафика:



Шаги конфигурации

1. Настройка ProviderNetwork с VLAN субинтерфейсами

Важно: необходимо использовать VLAN субинтерфейсы.

Настройте сеть Kube-OVN Underlay для автоматического создания VLAN субинтерфейсов:

```
apiVersion: kubeovn.io/v1
kind: ProviderNetwork
metadata:
  name: provider
spec:
  defaultInterface: underlay0.341
  autoCreateVlanSubinterfaces: true  # Автоматически создаёт VLAN субинте
рфейсы (например, underlay0.341), если существует только родительский инт
ерфейс (underlay0)

---
apiVersion: kubeovn.io/v1
kind: Vlan
metadata:
  name: ovn-vlan
spec:
  id: 0    # Используется 0, так как autoCreateVlanSubinterfaces создаёт
VLAN субинтерфейс (underlay0.341), который обрабатывает VLAN-тегирование,
а не Kube-OVN напрямую
  provider: provider
status:
  subnets:
  - ovn-default
```

> ⚠️ **Внимание**: При изменении ресурсов `ProviderNetwork` или `Vlan` по отдельности сетевое подключение Underlay будет прервано. Сеть восстановится только после полной настройки и синхронизации обоих ресурсов. Планируйте изменения конфигурации в окна обслуживания для минимизации прерывания сервиса.

2. Настройка параметров контроллера Kube-OVN

Настройте контроллер Kube-OVN с необходимыми параметрами для работы LoadBalancer:

Через веб-консоль:

1. Перейдите в **Administrator** > **Marketplace** > **Cluster Plugins**, затем найдите `ovn` для поиска плагина **Alauda Container Platform Networking for Kube-OVN**
2. В строке плагина нажмите меню действий (вертикальные ⋮) и выберите **Update** для открытия диалога конфигурации
3. Установите следующие параметры:
   - **Skip CT for Dst LPort IPs**: **No**
   - **Enable OVN LB Local**: **Yes**

3. Настройка функции внешних адресов подсети Underlay

Отредактируйте подсеть Underlay, чтобы зарезервировать диапазон IP для использования LoadBalancer:

> **Важно**: IP-адреса пула внешних адресов должны находиться в подсети Underlay.

Измените параметр подсети Underlay `spec.enableExternalLBAddress: true`:

```
apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
  name: underlay-subnet
spec:
  enableExternalLBAddress: true        # Указывает, что в этой подсети ест
ь диапазон IP для VIP LB сервиса
  excludeIps:
  - 10.180.208.200..10.180.208.220     # Резервирование диапазона IP для п
ула внешних адресов
```

4. Создание внешнего пула адресов MetalLB

```yaml
# underlay-ippool.yaml
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: acp-underlay-pool
  namespace: metallb-system
spec:
  addresses:
    - 10.180.208.200-10.180.208.220  # Диапазон IP подсети Underlay
  avoidBuggyIPs: true
  autoAssign: true
---
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: acp-underlay-pool
  namespace: metallb-system
spec:
  ipAddressPools:
    - acp-underlay-pool
  interfaces:
    - br-provider  # Опционально: интерфейс для ARP-рассылки; используйте мостовой интерфейс (br-*) вместо физического
  nodeSelectors: []
```

Примените пул адресов:

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

5. Создание примерного приложения и сервиса LoadBalancer

```yaml
# application-with-loadbalancer.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend-app
  labels:
    app: backend
spec:
  replicas: 3
  selector:
    matchLabels:
      app: backend
  template:
    metadata:
      labels:
        app: backend
    spec:
      containers:
        - name: backend
          image: nginx:1.25
          ports:
            - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: backend-lb-service
  # Для использования конкретного IPPool: добавьте аннотацию `metallb.io/address-pool: acp-underlay-pool`
  # Для использования фиксированного IP: задайте `spec.loadBalancerIP: 10.180.208.201`
spec:
  type: LoadBalancer
  externalTrafficPolicy: Local  # **ВАЖНО**: необходимо для сохранения исходного IP и прямой маршрутизации на Pod
  selector:
    app: backend
  ports:
    - port: 80
      targetPort: 80
```

Разверните приложение:

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

6. Проверка конфигурации

```
# Проверка статуса сервиса
kubectl get svc backend-lb-service -o wide

# Тест внешнего доступа
curl http://10.180.208.200
```

7. Миграция существующих сервисов

Для существующих сервисов, использующих старый пул адресов (только подсеть узла), можно выполнить миграцию на новый пул адресов Underlay:

```
# Добавить аннотацию для миграции существующего сервиса
kubectl annotate service <existing-service-name> metallb.io/address-pool=
acp-underlay-pool --overwrite

# Проверить, что сервис получил новый IP из пула Underlay
kubectl get svc <existing-service-name> -o wide
```

Для **новых сервисов** добавляйте аннотацию напрямую:

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

```
# Проверка статуса сервиса
kubectl get svc backend-lb-service -o wide

# Тест внешнего доступа
curl http://10.180.208.200
```

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.
- Должен быть набор машин, доступных по IP для kube-ovn-controller каждого кластера, чтобы развернуть контроллеры для межкластерного соединения.
- Для каждого кластера должен существовать набор машин, доступных по IP для kube-ovn-controller, 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 многозвенных Kube-OVN

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

Deploy Deployment

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

Шаги выполнения

1. Выполните следующую команду на Master-узле кластера для получения скрипта 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:

```
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 командой:

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

Развертывание с помощью Podman и Containerd

1. Выберите **три или более узлов в любом кластере** для развертывания Interconnected Controller. В данном примере подготовлено три узла.
2. Выберите любой узел в качестве Лидера и выполните команды в соответствии с выбранным методом развертывания.
   Примечание: Перед настройкой проверьте наличие каталога ovn в `/etc`. Если его нет, создайте командой `mkdir /etc/ovn`.

   - **Команды для развертывания в контейнерах**
     Примечание: Выполните `podman images | grep ovn` для получения адреса образа Kube-OVN.

     - Команда для узла Лидера:

```
podman run \
--name=ovn-ic-db \
-d \
--env "ENABLE_OVN_LEADER_CHECK=false" \
--network=host \
--restart=always \
--privileged=true \
-v /etc/ovn/:/etc/ovn \
-v /var/run/ovn:/var/run/ovn \
-v /var/log/ovn:/var/log/ovn \
-e LOCAL_IP="<IP-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:

```
podman run \
--name=ovn-ic-db \
-d \
--env "ENABLE_OVN_LEADER_CHECK=false" \
--network=host \
--restart=always \
--privileged=true \
-v /etc/ovn/:/etc/ovn \
-v /var/run/ovn:/var/run/ovn \
-v /var/log/ovn:/var/log/ovn \
-e LOCAL_IP="<IP-адрес текущего узла>" \   ## Например: -e LOCAL_IP="192.168.39.24"
-e LEADER_IP="<IP-адрес узла Лидера>" \  ## 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 развертывания с Containerd**

  Примечание: Выполните `crictl images | grep ovn` для получения адреса образа Kube-OVN.

- Команда для узла Лидера:

```
ctr -n k8s.io run \
-d \
--env "ENABLE_OVN_LEADER_CHECK=false" \
--net-host \
--privileged \
--mount="type=bind,src=/etc/ovn/,dst=/etc/ovn,options=rbind:rw" \
--mount="type=bind,src=/var/run/ovn,dst=/var/run/ovn,options=rbin
d:rw" \
--mount="type=bind,src=/var/log/ovn,dst=/var/log/ovn,options=rbin
d:rw" \
--env="NODE_IPS=<IP-адреса всех узлов через запятую>" \   ## Напри
мер: --env="NODE_IPS="192.168.178.97,192.168.181.93,192.168.177.19
2""
--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   ## Нап
ример: registry.alauda.cn:60080/acp/kube-ovn:v1.9.25 ovn-ic-db bas
h 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=rbin
d:rw" \
--mount="type=bind,src=/var/log/ovn,dst=/var/log/ovn,options=rbin
d:rw" \
--env="NODE_IPS=<IP-адреса всех узлов через запятую>" \   ## Напри
мер: --env="NODE_IPS="192.168.178.97,192.168.181.93,192.168.177.19
2"" \
--env="LOCAL_IP=<IP-адрес текущего узла>" \   ## Например: --env
="LOCAL_IP="192.168.181.93""
--env="LEADER_IP=<IP-адрес узла Лидера>" \   ## Например: --env="L
EADER_IP="192.168.178.97""
<адрес репозитория образов> ovn-ic-db bash start-ic-db.sh   ## Нап
ример: registry.alauda.cn:60080/acp/kube-ovn:v1.9.25 ovn-ic-db bas
h start-ic-db.sh
```

Развертывание контроллера межкластерного соединения в глобальном кластере

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

Примечание: Для корректной работы ConfigMap с именем ovn-ic в global запрещено изменять. Если необходимо изменить параметры, удалите ConfigMap и заново корректно настройте его 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 узло
в, где расположен контроллер межкластерного соединения, в данном случае л
окальные IP трех узлов с контроллером
  ic-nb-port: "6645"            # nb порт Cluster Interconnect Controlle
r, по умолчанию 6645
  ic-sb-port: "6646"            # sb порт Cluster Interconnect Controlle
r, 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 **присоединяемые подсети** в кластере не должны конфликтовать с любым сегментом кластера 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 шлюзовой узел для кластера.
6. Нажмите **Join**.

Соответствующие операции

Обновление информации о шлюзовом узле подключенного кластера

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

Порядок действий

1. В левой навигационной панели нажмите **Clusters** > **Cluster of clusters**.
2. Нажмите на ***название 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 шлюзовой узел для кластера.
6. Нажмите **Update**.

Выход из межкластерного соединения

Кластер, присоединившийся к группе межкластерного соединения, выходит из межкластерного соединения, при этом отключается доступ подов кластера к подам 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 межкластерного соединения

Если кластер удаляется без выхода из межкластерного соединения, на контроллере могут остаться некоторые остаточные данные. При попытке повторного создания кластера на этих узлах и присоединения 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 xxxxx
x"}
```

Шаги выполнения

1. Выйдите из межкластерного соединения для кластера, который собираетесь присоединить.
2. Выполните скрипт очистки в контейнере или поде.
   Скрипт очистки можно выполнить напрямую либо в контейнере ovn-ic-db, либо в поде ovn-ic-controller. Выберите один из следующих способов:
   Способ 1: Выполнение в контейнере ovn-ic-db

   - Зайдите 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 поде ovn-ic-controller

- Зайдите в под 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 межкластерного соединения

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

Шаги выполнения

1. Выйдите из межкластерного соединения. Существуют 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. Зайдите в Pod Лидера ovn-central 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 командой:

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

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

   - Команды для Podman:

   ```
   podman stop ovn-ic-db
   podman 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 командой:

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

Настройка высокой доступности шлюза кластера

Чтобы настроить высокую доступность шлюза кластера после присоединения к межкластерному соединению, выполните следующие шаги:

1. Войдите в кластер, который нужно преобразовать в шлюз с высокой доступностью, и выполните команду для изменения поля `enable-ic` на `false`.
   Примечание: Изменение поля `enable-ic` на `false` прервет межкластерное соединение до повторного установки значения `true`.

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

2. Измените конфигурацию шлюзового узла, обновив поле `gw-nodes`, разделяя узлы запятыми, и 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: az1
  enable-ic: "true"
  gw-nodes: 192.168.188.234,192.168.189.54
  ic-db-host: 192.168.178.97
  ic-nb-port: "6645"
  ic-sb-port: "6646"
kind: ConfigMap
metadata:
  creationTimestamp: "2023-06-13T08:01:16Z"
  name: ovn-ic-config
  namespace: kube-system
  resourceVersion: "99671"
  uid: 6163790a-ad9d-4d07-ba82-195b11244983
```

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

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

# Пример вывода. В данном случае значения 100 и 99 — это приоритеты, че
м выше значение, тем выше приоритет соответствующего шлюзового узла.
az1-ts-71292a21-131d-492a-9f0c-0611af458950 100
az1-ts-1de7ee15-f372-4ab9-8c85-e54d61ea18f1 99
```

Настройка Egress Gateway

Содержание

Описание Egress Gateway

Egress Gateway используется для контроля доступа к внешней сети для Pod с группой статических адресов и обладает следующими возможностями:

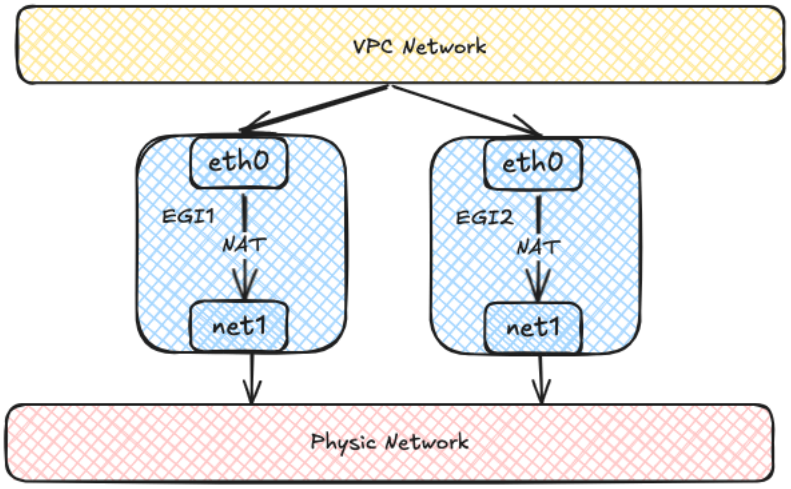
- Обеспечивает высокую доступность Active-Active через ECMP, позволяя горизонтально масштабировать пропускную способность
- Реализует быстрое переключение при сбое (<1c) с помощью BFD
- Поддерживает IPv6 и dual-stack
- Позволяет тонко управлять маршрутизацией через NamespaceSelector и PodSelector
- Обеспечивает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

При этом Egress Gateway имеет следующие ограничения:

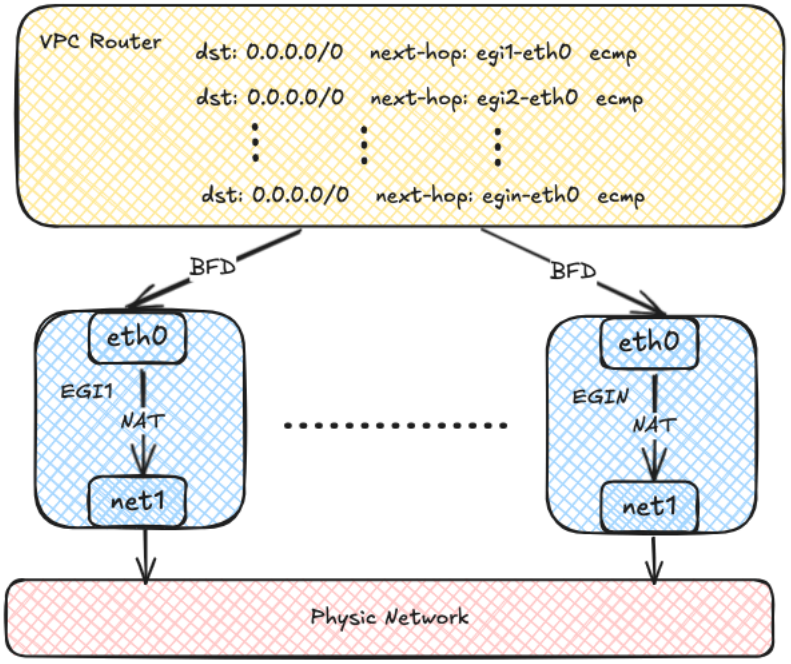
- Использует macvlan для базового сетевого подключения, требуя поддержки Underlay со стороны физической сети
- В режиме многократных экземпляров Gateway требуется несколько Egress IP
- В настоящее время поддерживается только SNAT; EIP и DNAT не поддерживаются
- В настоящее время не поддерживается запись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

Детали реализации

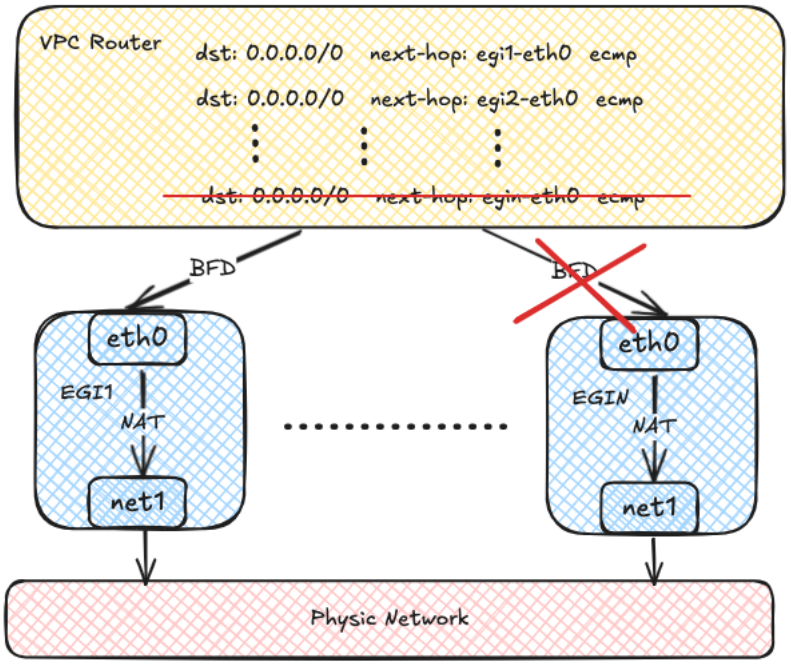
Каждый Egress Gateway состоит из нескольких Pod с несколькими сетевыми интерфейсами. Каждый Pod имеет два сетевых интерфейса: один подключается к виртуальной сети для связи внутри VPC, а другой подключается к базовой физической сети через Macvlan для связи с внешней сетью. Трафик виртуальной сети в конечном 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 необходимо получить доступ к внешней сети, OVN использует хеширование исходного адреса для передачи трафика на несколько адресов экземпляров Egress Gateway, обеспечивая балансировку нагрузки. По мере увеличения 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 сбоя.



Предварительные требования

Должна быть установлена Alauda Container Platform Networking for Multus **до** использования Egress Gateway.

Для установки *Alauda Container Platform Networking for Multus* обратитесь к разделу Installing Multus CNI.

Использование

Создание Network Attachment Definition

Egress Gateway использует несколько NIC 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, предоставляемым 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 (4)
  gateway: 172.17.0.1 (5)
  excludeIps: (6)
    - 172.17.0.2..172.17.0.10
```

(1) Хост-интерфейс, подключающийся к внешней сети.

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

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

(4) CIDR внешней сети.

(5) Шлюз внешней сети.

(6) Диапазон IP, исключённый из автоматического распределения. Подробнее см. в разделе Example Subnet custom resource (CR) with Kube-OVN Overlay Network.

> **TIP**
>
> Вы можете создать Network Attachment Definition с любым CNI-плагином для доступа к соответствующей сети. Альтернативой приведённому примеру является использование подсети Kube-OVN underlay вместо macvlan.

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)
  resources: (4)
    requests:
      cpu: 100m
      memory: 128Mi
    limits:
      cpu: 200m
      memory: 256Mi
      ephemeral-storage: 2Gi
  nodeSelector: (5)
    - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
            - kube-ovn-worker
            - kube-ovn-worker2
  tolerations: (6)
    - key: node-role.kubernetes.io/control-plane
      operator: Exists
      effect: NoSchedule
  selectors: (7)
    - namespaceSelector:
        matchLabels:
          kubernetes.io/metadata.name: default
  policies: (8)
    - snat: true (9)
      subnets: (10)
        - subnet1
    - snat: false
      ipBlocks: (11)
        - 10.18.0.0/16
```

(1) Namespace, в котором создаются экземпляры VPC Egress Gateway.

(2) Количество реплик экземпляров VPC Egress Gateway.

(3) Внешняя подсеть, подключающаяся к внешней сети.

(4) Запросы и лимиты ресурсов для каждого экземпляра VPC Egress Gateway. Если не указано, применяются значения по умолчанию, определённые в контроллере VPC Egress Gateway.

(5) Node selector для планирования экземпляров VPC Egress Gateway.

(6) Tolerations для планирования экземпляров VPC Egress Gateway.

(7) Namespace selector и Pod selector для выбора Pod, которые получают доступ к внешней сети через VPC Egress Gateway.

(8) Политики для VPC Egress Gateway, включая SNAT и применяемые подсети/ipBlocks.

(9) Включение SNAT для политики.

(10) Подсети, к которым применяется политика.

(11) IP-блоки, к которым применяется политика.

Данный ресурс создаёт VPC Egress Gateway с именем *gateway1* в namespace default, и следующие Pod будут получать доступ к внешней сети через подсеть *macvlan1*:

- Pod в namespace default
- Pod в подсети *subnet1*

- Pod с IP в CIDR *10.18.0.0/16*

> **NOTE**
>
> Pod, соответствующие *.spec.selectors*, 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 IP
S      WORKING NODES         AGE
gateway1   ovn-cluster   1          false         macvlan1          Completed   true    ["10.16.0.12"]   ["172.17.0.
11"]   ["kube-ovn-worker"]   82s
```

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   READ
INESS GATES
gateway1-b9f8b4448-76lhm    1/1     Running   0          4m48s   10.16.0.12   kube-ovn-worker   <none>           <non
e>
```

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 304
81, 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 3048
1, 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 2670
1, 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 267
01, 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
```

Маршрутизирующие политики 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 Pod, указанных в *.spec.policies*.

(2) Политика Logical Router, используемая VPC Egress Gateway для перенаправления трафика от Pod, 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   READI
NESS 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/Subnet.

(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.

После этого 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 объявления сбоя.

В этом примере создаётся VPC Egress Gateway с именем *gateway2* с 2 репликами и включённым BFD. Если один из экземпляров выйдет из строя, сессия BFD упадёт, и OVN быстро обнаружит сбой и прекратит передачу трафика на неработающий экземпляр. Весь трафик будет перенаправлен на исправный экземпляр, обеспечивая непрерывный доступ к внешней сети.

Время переключения зависит от конфигурации BFD. Его можно вычислить по формуле: *время переключения = (multiplier + 1) * max(minRX, minTX)*. В данном примере обнаружение сбоя и перенаправление трафика займёт около 500–600 миллисекунд.

> **NOTE**
>
> Существующие соединения могут быть прерваны во время переключения, потребуется их восстановление. Однако новые соединения не будут затронуты и могут устанавливаться нормально.

Для просмотра информации о VPC Egress Gateway:

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

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

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

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

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

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
```

NOTE

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		Примеры
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.		169.255.25 fdff::1 169.255.25
nodeSelector	matchLabels	object	Да	-	Селекторы меток для выбора узлов, на которых работает BFD Port. BFD Port связывает 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	Описание
vpc			string	Да	Имя default VPC (ovn-cluster)	Имя VPC.
replicas			integer/int32	Да	1	Количество реплик.
prefix			string	Да	-	Неизменяемый префикс им
image			string	Да	-	Образ, используемый для р
internalSubnet			string	Да	Имя default подсети внутри VPC.	Имя подсети для доступа к
externalSubnet				Нет	-	
internalIPs			string array	Да	-	IP-адреса для доступа к вну Поддерживаются IPv4, IPv6 Количество указанных IP не Рекомендуется указать коли избежания крайних случаев
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 трафика по селекторам namespace и Pod. SNAT/MASQUERADE
		matchExpressions	object array	Да	-	

Поля			Тип	Необязательно	Значение по умолчанию	Описание
						будет применён к выбранным Pod.
	podSelector	matchLabels	object	Да	-	
		matchExpressions	object array	Да	-	
nodeSelector	matchLabels		object	Да	-	Node selector для выбора узлов, на которых развёртывается workload. Workload (Deployment/Pod) будет запущен на выбранных узлах.
	matchExpressions		object array	Да	-	
	matchFields		object array	Да	-	
tolerations	key		string	Да	-	Tolerations для планирования экземпляров VPC Egress Gateway.
	operator		string	Да	Equal	
	value		string	Да	-	

Поля		Тип	Необязательно	Значение по умолчанию	Описание
	effect	string	Да	-	
	tolerationSeconds	integer	Да	-	
trafficPolicy		string	Да	Cluster	**Эффективно только при в** Допустимые значения: *Clus* При установке в *Local* исход на экземпляр VPC Egress G узле, если он доступен. Если экземпляр недоступен другие экземпляры.

Примечания

Любое действие, приводящее к удалению/пересозданию экземпляра Egress Gateway, может вызвать временный сбой сети исходящего трафика, включая, но не ограничиваясь:

1. Изменение количества реплик;
2. Изменение некоторых конфигураций, например, внутренних/внешних IP, node selector, конфигурации BFD и т.д.;
3. Обновление/понижение версии Kube-OVN, если *.spec.image* не указано;
4. Ручное удаление Pod экземпляра Egress Gateway.

Дополнительные ресурсы

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

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 **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 attachnet подсеть 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 VLAN CR, который 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 выполните команду `kubectl exec pod1 -- ip a`, чтобы просмотреть IP-адреса Pod.

Проверка создания двух сетевых интерфейсов

Используйте следующую команду, чтобы убедиться, что два сетевых интерфейса созданы успешно:

```
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, способ такой же, как 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)**: Базовый способ аналогичен основному интерфейсу, с тем отличием, что `ovn` в ключе аннотации заменяется на соответствующий провайдер 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 IPPool

IPPool — это более детализированная единица управления IPAM, чем Subnet.
Вы можете разделить сегмент подсети на несколько единиц с помощью IPPool, и каждая единица привязывается к одному или нескольким namespace.

Содержание

Инструкции

Создание IPPool

Ниже приведён пример:

```yaml
apiVersion: kubeovn.io/v1
kind: IPPool
metadata:
  name: pool-1
spec:
  subnet: ovn-default (1)
  ips: (2)
  - "10.16.0.201"
  - "10.16.0.210/30"
  - "10.16.0.220..10.16.0.230"
  namespaces: (3)
  - ns-1
```

(1) Подсеть, к которой принадлежит IP pool.

(2) Диапазоны IP. Поддерживаемые форматы: *<IP>*, *<CIDR>* и *<IP1>..<IP2>*. Поддерживаются как IPv4, так и IPv6.

(3) Опциональные namespace, к которым привязан IP pool. Подам в привязанном namespace будут выделяться IP только из привязанных pool-ов, а не из других диапазонов подсети(ей).

Использование IPPool

Чтобы назначать IP случайным образом из IP pool, просто привяжите IP pool к нужному namespace(ам).
При создании Pod-ов в привязанном namespace их IP будут выделяться из соответствующего IP pool(ов).

> **NOTE**
>
> Перед привязкой IP pool к namespace убедитесь, что к namespace привязана только подсеть, которой принадлежит IP pool.

Вы также можете назначить IP pool Pod-ам через аннотацию:

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

Для рабочих нагрузок используйте аннотацию в шаблоне Pod в Deployment, StatefulSet и т.д.:

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

Меры предосторожности

1. Для обеспечения совместимости с функцией *Fixed Addresses* имя IP pool не может быть IP-адресом.

2. Разрешены IP-адреса вне диапазона подсети, однако такие IP не будут иметь эффекта.
3. Разные IP pool, принадлежащие одной подсети, не могут иметь пересекающиеся диапазоны IP.
4. Поле `.spec.ips` можно обновлять в любое время. Все изменения вступают в силу немедленно.
5. IP pool наследует зарезервированные IP подсети. При случайном выделении IP из IP pool зарезервированные IP в диапазоне pool-а будут пропускаться.
6. При случайном выделении IP из подсети исключаются IP-диапазоны всех IP pool в этой подсети.
7. Несколько IP pool могут быть привязаны к одному namespace.

Настройка MTU

В Kube-OVN параметр MTU (Maximum Transmission Unit) имеет решающее значение для обеспечения оптимальной производительности сети и предотвращения фрагментации пакетов.
MTU определяет максимальный размер пакета, который может быть передан по сети.

По умолчанию Kube-OVN определяет MTU физического сетевого интерфейса и устанавливает MTU для виртуальных сетевых интерфейсов соответственно.
Однако существуют сценарии, когда может потребоваться настроить параметры MTU для компонентов сети Kube-OVN вручную.

Содержание

Поведение MTU по умолчанию

Kube-OVN автоматически определяет MTU физического сетевого интерфейса на хост-машине и устанавливает MTU для интерфейсов Pod и OVS соответственно.

В оверлейных сетях Kube-OVN уменьшает MTU, чтобы учесть накладные расходы на инкапсуляцию VXLAN или Geneve.

Расчет MTU производится следующим образом:

Тип инкапсуляции	Версия IP туннеля	Расчет MTU
Geneve	IPv4	MTU физического сетевого интерфейса - 100
	IPv6	MTU физического сетевого интерфейса - 120
VXLAN	IPv4	MTU физического сетевого интерфейса - 50
	IPv6	MTU физического сетевого интерфейса - 70

В андерлейных сетях MTU устанавливается равным MTU физического сетевого интерфейса.

Настройка параметров MTU

Параметры MTU можно настроить глобально для оверлейных сетей или для каждой подсети отдельно.

WARNING

Неправильная настройка MTU может привести к проблемам с производительностью сети, включая потерю пакетов и фрагментацию.
Перед изменением убедитесь, что параметры MTU совместимы с вашей базовой сетевой инфраструктурой.

Критический момент при увеличении MTU

При увеличении MTU с меньшего значения на большее необходимо перезапустить все Pods, чтобы новые параметры MTU применились равномерно по всему кластеру.

Почему это важно?

Внутренние порты OVS (например, `ovn0`) автоматически принимают в качестве MTU **минимальное значение MTU** среди всех интерфейсов, подключенных к мосту `br-int`. Это может привести к следующей проблеме:

1. Вы задаёте большее значение MTU для новых Pods
2. Некоторые существующие Pods продолжают использовать исходный меньший MTU
3. Интерфейс `ovn0` сохраняет меньший MTU из-за правила минимального MTU
4. Трафик от Pods с большим MTU теряется, если пакеты превышают MTU интерфейса `ovn0`

Решение: После увеличения MTU необходимо пересоздать все Pods, чтобы обеспечить согласованную конфигурацию MTU по всей сетевой цепочке.

NOTE

Изменения конфигурации MTU вступают в силу только для вновь созданных Pods.
Существующие Pods сохраняют свои исходные настройки MTU до тех пор, пока не будут пересозданы.
Рекомендуется планировать перезапуск Pods при изменении параметров MTU.

Глобальная настройка MTU

Для установки глобального MTU для оверлейных сетей можно изменить параметр команды в DaemonSet `kube-ovn-cni`.
Например, чтобы установить глобальный MTU равным 1400, выполните редактирование DaemonSet следующим образом:

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

Затем добавьте или обновите параметр `--mtu=1400` в аргументах контейнера:

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

NOTE

Глобальная настройка MTU влияет только на оверлейные сети.

Для андерлейных сетей MTU по умолчанию остается равным MTU физического сетевого интерфейса.

Настройка MTU для каждой подсети

Также можно задать MTU для отдельных подсетей, указав поле `mtu` в конфигурации подсети.

Например, чтобы установить MTU равным 1450 для конкретной подсети, выполните команду:

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

Эта настройка переопределит глобальное значение MTU для данной подсети.

Настройка Endpoint Health Checker

Содержание

Overview

Endpoint Health Checker — это плагин кластера, предназначенный для мониторинга и управления состоянием здоровья сервисных эндпоинтов в кластере k8s. Он автоматически удаляет нездоровые эндпоинты из сервиса, чтобы обеспечить маршрутизацию трафика только к здоровым экземплярам, повышая общую надежность и доступность сервиса.

Key Features

- **Автоматический мониторинг здоровья**: Непрерывно отслеживает состояние здоровья сервисных эндпоинтов в кластере k8s
- **Интеграция с балансировщиком нагрузки**: Автоматически удаляет нездоровые эндпоинты из сервиса
- **Доступность сервиса**: Обеспечивает направление трафика только к здоровым, доступным эндпоинтам
- **Быстрое переключение**: Снижает время переключения эндпоинтов с 40 с до 10 с при отключении узлов

Installation

Установка через Marketplace

1. Перейдите в **Administrator** > **Marketplace** > **Cluster Plugins**.
2. Найдите в списке плагинов "**Alauda Container Platform Endpoint Health Checker**".
3. Нажмите **Install** для открытия страницы конфигурации установки.
4. В диалоговом окне настройки развертывания можно при необходимости задать следующие параметры:

Параметр	Описание
Node Selectors	Настройте селекторы меток, чтобы указать, на каких узлах должны запускаться компоненты Endpoint Health Checker. Нажмите **Add** для добавления нескольких пар ключ-значение меток.

Параметр	Описание
Node Tolerations	Настройте толерантности, чтобы разрешить запуск компонентов Endpoint Health Checker на узлах с определёнными taints. Нажмите **Add** для добавления нескольких толерантностей с Key, Value и Type.

5. Нажмите **Install** для развертывания плагина.
6. Дождитесь, пока статус плагина изменится на "**Ready**".

How It Works

Механизм проверки здоровья

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

Основной функционал

Endpoint Health Checker работает следующим образом:

1. **Обнаружение сервисов**: Определяет сервисы и поды, настроенные для мониторинга здоровья в кластере.
2. **Мониторинг здоровья подов**: Отслеживает состояние readiness и liveness probe подов, поддерживающих сервисные эндпоинты.
3. **Активные проверки здоровья**: Выполняет активные проверки здоровья с использованием настраиваемых критериев:
   - **Проверки TCP-соединений**: Устанавливает TCP-соединения для проверки доступности портов.
4. **Управление эндпоинтами**: Автоматически удаляет нездоровые эндпоинты из списков сервисных эндпоинтов, чтобы предотвратить маршрутизацию трафика к неработающим экземплярам.

Процесс проверки здоровья

Процесс проверки здоровья включает:

- **Интеграция с probe**: Использует результаты readiness и liveness probe Kubernetes в качестве начальных индикаторов здоровья.
- **Сетевое соединение**: Отправляет TCP-пакеты на порты целевых эндпоинтов для проверки доступности.
- **Проверка ответа**: Оценивает статус ответа, время и содержимое для определения состояния эндпоинта.
- **Автоматическое переключение**: Удаляет неотвечающие или сбойные эндпоинты из списков сервисных эндпоинтов.

Улучшение производительности

- **Предыдущий метод**: Опирался на обнаружение heartbeat kubelet с задержкой до 40 секунд.
- **Текущий метод**: Активная проверка здоровья эндпоинтов с временем обнаружения и переключения 10 секунд.
- **Преимущество**: Значительно повышает доступность сервиса при сбоях узлов в средах ALB + MetalLB.

How To Activate

Проверка здоровья может быть активирована двумя способами:

Аннотация на уровне пода (рекомендуется)

Для ALB

установите аннотацию `alb.cpaas.io/pod-annotations` в `ALB2`

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

Для IngressNginx

1. Установите ingress-nginx
2. Задайте `podAnnotations` в `.spec.controller.podAnnotations` объекта `IngressNginx`.

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

Для EnvoyGateway

1. Установите envoy-gateway-operator
2. Задайте `annotations` в `.spec.provider.kubernetes.envoyDeployment.patch.value.spec.template.metadata.annotations` объекта `EnvoyProxy`.

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

Для пользовательского Deployment

установите `annotations` в `.spec.template.metadata.annotations` объекта `Deployment`

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

readinessGates на уровне пода (устаревший способ)

Настройте readinessGates в спецификации пода для старых версий:

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

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

Uninstallation

Для удаления 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.

Задачи для ALB

Содержание

Как задать NodeSelector и Tolerations для alb-operator

обновите ресурсы `deployment`

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

Как задать NodeSelector и Tolerations для alb

обновите ресурсы `alb`

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

Устранение неполадок

Как решить проблемы межуз Определение причины ошибки

Как решить проблемы межузловой связи 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 связи между узлами в оверлейной сети Kube-OVN. Конкретные решения следующие:

- **Решение 1: Обновить версию ядра**. Рекомендуется обновить версию ядра до 4.19.90-25.16.v2101 или выше.
- **Решение 2: Отключить Checksum Offload**. Если невозможно сразу обновить версию ядра и возникают проблемы с межузловой связью, можно отключить Checksum Offload для физической сетевой карты с помощью следующей команды.

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

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 сервис
а не найден endpoint
BackendError : read xxx byte data from backend # это означает, что backen
d вернул ответ, ошибка не вызвана alb.
InvalidUpstream : no rule match # это означает, что запрос не соответству
ет ни одному правилу, поэтому alb возвращает 404.
```